UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

ANÁLISIS DE SOLUCIONES PARA REDES DE COMUNICACIONES CABLEADAS PARA SU USO EN DISPOSITIVOS IOT

Marco Pedro Cuevas Gómez
Tutor: Fernando Jesús López Colino

Julio 2019

ANÁLISIS DE SOLUCIONES PARA REDES DE COMUNICACIONES CABLEADAS PARA SU USO EN DISPOSITIVOS IOT

AUTOR: Marco Pedro Cuevas Gómez
TUTOR: Fernando Jesús López Colino

Grupo: HCTLab
Dpto. TEC
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Julio de 2019

Resumen (castellano)

Actualmente, vivimos en un mundo impulsado por la tecnología, en el que cada vez es más frecuente escuchar ideas que en un pasado fueron de ciencia ficción y que en la actualidad son ya una realidad.

Este Trabajo Fin de Grado trata de dar una visión general sobre las diferentes tecnologías por cable que existen en la actualidad, para poder realizar una comunicación a nivel global de cualquier dispositivo existente. Con ello se pretende, dotar de comunicación e inteligencia a cualquier dispositivo electrónico que dispongamos, dando lugar al término conocido como Internet de las cosas (IoT).

Específicamente, este Trabajo de Fin de Grado llevará a cabo la comunicación de dos sistemas empotrados a través del protocolo de líneas de potencia (PLC), realizando una transmisión y recepción de datos.

Más detalladamente, los sistemas empotrados serán dos Raspberry Pi que se comunicarán entre ellas mediante corriente alterna. Mientras, uno de los sistemas empotrados realizará adicionalmente una conexión I^2C con varios elementos con diferentes funcionalidades y especificaciones. Adicionalmente, el otro sistema empotrado albergará un servidor web, capaz de procesar las peticiones entrantes a través de una página web, que servirá para interactuar de manera sencilla con toda la funcionalidad implementada en el proyecto.

Con todo lo realizado en el Trabajo de Fin de Grado, se valora el uso de la línea eléctrica de potencia como alternativa a considerar en el despliegue de una red IoT, mediante una arquitectura híbrida de comunicación centrada en dicha alternativa. Esta arquitectura, incluye comunicación TCP/IP sobre Ethernet, PLC e I^2C.

Finalmente, queremos destacar que el uso de la línea eléctrica facilitaría en muy amplia medida la evolución tecnológica en el área de IoT.

Abstract (English)

Currently, we live in a word driven by technology, in which it is more common to listen ideas that in the past were science-fiction and that are now a reality.

This Bachelor Thesis try to give an overview about different wired technologies that exist today, in order to make a global communication of any existing device. Therefore, we want to provide communication and intelligence to any electronic device we have, giving rise to the term known as the Internet of Things (IoT).

Specifically, this Bachelor Thesis will carry out the communication of two embedded systems through the power line, carrying out a transmission and reception of data, which will carry out one of the two embedded systems.

In more detail, the embedded systems will be two Raspberry Pi that will communicate with each other through the power line, making use of the serial port. Meanwhile, one of the embedded systems will additionally make an I^2C connection with several elements with different functionalities and specifications. In addition, the other embedded system will accommodate a web server, capable of processing incoming request though a web page, which will help to interact in a simple way with all the functionality implemented in the project.

With everything done in the Bachelor Thesis, the use of power electric line is valued as alternative to considering the deployment of an IoT network, through a hybrid communication architecture focused on sail alternative. This architecture includes TCP/IP communication over Ethernet, PLC and I^2C.

Finally, we would like to point out that the use of power line would greatly facilitate technological evolution in the area of the Internet of Things.

Palabras clave (castellano)

Internet de las cosas, sistema empotrado, Raspberry Pi, puerto serie, I^2C y línea eléctrica.

Keywords (inglés)

Internet of things, embedded system, Raspberry Pi, serial port, I^2C and power line,

Agradecimientos

En esta sección de mi trabajo de fin de grado me gustaría agradecer a todas las personas que me han apoyado y dado fuerza en momentos de debilidad y que siempre han apostado por mí para que yo pueda llegar a ser lo que soy ahora. Este trabajo de fin de grado no habría sido posible realizarlo sin todas ellas.

En especial quería darle un gran agradecimiento a Fernando López, mi tutor de este trabajo, al que le pareció una gran idea que cooperásemos juntos y el cual siempre ha estado atento a mis peticiones y consultas a través del mail.

También del profesorado me gustaría darle las gracias a Miren Idoia, una profesora de la universidad que ha sido una gran madre para mí, ayudándome siempre y respaldándome en los momentos tensos y de dificultad, dándome siempre sus mejores consejos. Y a Eloy Anguiano, que siempre ha estado para recibirme explicándome las cosas que más me costaban entender de la carrera.

Por otra parte, me gustaría agradecer a todos mis amigos, familiares y novia, toda la confianza y motivación que han depositado en mí a lo largo de la carrera, aguantándome en situaciones de mayor estrés y apoyándome siempre.

De todo corazón os deseo lo mejor, muchas gracias por ser parte de mi vida.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

1 Introducción

1.1 Motivación

Este proyecto surge de la necesidad actual de perseguir una comunicación global de todos los dispositivos, haciendo uso de las tecnologías más avanzadas. La motivación es muy clara, dotar a un sistema empotrado de la capacidad para usar la red eléctrica de potencia como medio de comunicación. De esta manera, podríamos dotar a un edificio entero de comunicación, que posteriormente podríamos gestionar a través de Internet.

Esta capacidad de comunicar todos los dispositivos bajo una misma red nos permite controlar todos los elementos enlazados a dicha red, que conectando uno de ellos a Internet nos permitiría hacer uso del término IoT.

Finalmente, demostraremos que no siempre lo más rápido es lo más eficiente y que puede resultar más útil realizar una conexión más lenta pero accesible a todo el mundo.

1.2 Objetivos

El principal objetivo de este proyecto es realizar la comunicación de dos sistemas empotrados mediante el uso de corriente alterna. Para ello, realizaremos una revisión exhaustiva sobre las diferentes tecnologías y su impacto actuales.

Para cumplir con el objetivo principal de nuestro proyecto, deberemos cumplir los siguientes objetivos:

- Integrar un diseño sobre un módulo de comunicaciones que utiliza línea eléctrica.
- Realizar la implementación del software de un módulo de comunicaciones para plataformas empotradas a través de la línea eléctrica.
- Comprender el protocolo utilizado por el módulo de comunicación KQ-330.
- Definir un nuevo protocolo para transmitir datos mediante la línea eléctrica de potencia de manera sencilla.

Nuestro objetivo final, será realizar una conexión completa tanto de transmisión como recepción de datos a través de la línea eléctrica, demostrando pues, que para determinadas situaciones en más útil y eficiente realizar una comunicación con otras tecnologías de transporte físico.

1.3 Organización de la memoria

La memoria consta de los siguientes capítulos:

- **Estado del arte**; en el que se realiza una investigación minuciosa sobre las tecnologías existentes actualmente, en las que analizamos sus ventajas frente a otras tecnologías. También, se analizarán las diferentes topologías de red que se conocen hoy en día, para saber cuál podría ser la mejor en nuestro proyecto.
- **Diseño**; en el cual realizaremos un mapa conceptual de la arquitectura de nuestro proyecto. También analizaremos los límites que abarca nuestro proyecto, dado que podría llegar a ser muy extenso. Por último, hablaremos de la funcionalidad que se implementará en el desarrollo.
- **Desarrollo**; en el que explicaremos las diferentes partes de las que se compone el proyecto, como se han implementado, como se han realizado sus conexiones, cuál es su estructura y en algunos casos, cual ha sido la configuración previa realizada para conseguir un correcto funcionamiento.
- **Integración, pruebas y resultados**; capítulo en el que realizaremos las verificaciones de código y hardware necesarias para un correcto funcionamiento de toda la funcionalidad implementada en el desarrollo.
- **Conclusiones y trabajo futuro**; en el que hablaremos de las propias conclusiones extraídas al finalizar el proyecto, así como posibles puntos de fallo y trabajo e implementaciones pendientes por realizar.

2 Estado del arte

Actualmente, existen múltiples tecnologías físicas que ayudan a conectar prácticamente cualquier dispositivo en una red de datos. Estas tecnologías han ido evolucionando para proporcionar una mayor comodidad y velocidad de transmisión.

Realizar una comunicación de varios dispositivos a través de Internet puede resultar muy útil si deseamos controlar diferentes acciones sin necesidad de estar en el lugar físicamente. Por ejemplo, podemos obtener los datos de humedad de un lugar para saber si es necesario regar o no, controlar la iluminación en lugares oscuros e incluso controlar la calefacción en lugares fríos, todo ello desde largas distancias.

2.1 Tecnologías de nivel físico [1]

Hoy en día, podemos destacar diversas opciones de medios físicos con los que podremos conectar nuestros dispositivos, dichos medios se conocen como medios de transmisión guiados.

2.1.1 Par trenzado

Es comúnmente conocido por ser usado antiguamente por las compañías telefónicas. Su principal característica es su bajo coste económico que lo hace especialmente interesante. Podemos encontrar dos tipos diferentes de cable de par trenzado:

- Par trenzado **sin blindaje** o UTP (Unshielded Twisted Pair), utilizado en la mayor parte de los edificios y telefonías móviles, al ser el más barato de ambos tipos y el más fácil de instalar y manipular.
- Par trenzado **con blindaje** o STP (Shielded Twisted Pair), envuelto por una malla metálica, reduce las interferencias electromagnéticas externas, proporcionando mayor velocidad de transmisión y por consiguiente mayor coste que el par trenzado sin blindaje.

También encontramos diversas versiones del cable de par trenzado que constituyen la evolución de dicho cable a lo largo de los años, conocidas como categorías.

2.1.2 Línea eléctrica [2]

Una de las mejores opciones a nivel económico es utilizar las instalaciones de energía eléctrica para transmitir datos. Pero su mayor desventaja es que al tener conectados múltiples dispositivos a la red eléctrica, estos generan ruido eléctrico a través de diversos rangos de frecuencia, lo que podría implicar pérdida de datos. El ruido generado, dependerá por lo tanto de la frecuencia a la que trabaje cada dispositivo y será muy variable. Por ejemplo, los electrodomésticos funcionan a frecuencias bajas. Este será nuestro medio guiado escogido para realizar nuestro proyecto y las ventajas podrían ser muy determinantes en especial en lugares en los que hacer llegar un cable de cobre no sea trivial.

2.1.3 Cable coaxial

Era utilizado principalmente para la transmisión de televisión y telefonía de larga distancia. Actualmente los servicios de televisión han cambiado y ahora la televisión utiliza la fibra óptica, puesto que la televisión se transmite por Internet. Ofrece una mayor distancia de comunicación que el cable de cobre de par trenzado, aunque peor velocidad de transmisión.

2.1.4 Fibra óptica

Es actualmente el medio físico de transmisión de datos más rápido, su principal desventaja es su alto coste en comparación al resto de medios de transmisión físicos. Podemos distinguir dos tipos, **multimodo** y **monomodo**. El multimodo es utilizado en distancias cortas, como en un edificio o campus. Es la que se lleva a los diferentes usuarios y más barata que la fibra monomodo. La fibra monomodo es utilizada para grandes distancias y generalmente son usadas en las redes troncales de las empresas telefónicas. Lo que realmente le da ventaja sobre el resto de las opciones, es su baja atenuación e interferencia, que gracias a estas dos características hacen que la velocidad de transmisión sea tan elevada.

2.2 Estándares y protocolos

A continuación, se hará una breve descripción de cada uno de los estándares y protocolos existentes en la actualidad, para posteriormente escoger uno para la implementación de nuestro proyecto.

2.2.1 Ethernet [3]

Es un estándar de las redes de área local (LAN), que especifica la capa física y la capa de enlace de datos del modelo OSI [4].

Conocido como el estándar IEEE 802.3. Utilizan una técnica llamada CSMA/CD (Carrier Sense Multiple Access with Collision Detection). Con CSMA, una estación que desee transmitir en un medio escuchará primero al medio para determinar si existe o no otra transmisión en curso. Y con CD, cuando se produzca una colisión (dos estaciones han realizado la transmisión a la vez), se realizará un envió de una pequeña señal de interferencia, para avisar al resto de estaciones de que se ha producido una colisión y se dejará de transmitir. Posteriormente, se utilizará una técnica conocida como espera exponencial binaria [5] y se volverá a intentar la transmisión. A nivel físico, utiliza la codificación denominada Manchester diferencial [6]. Esta codificación, posee tres estados: Transmisión de ‘0’ lógico, una señal de +0.85 V seguida de otra de -0.85 V; Transmisión de ‘1’ lógico, una señal de -0.85 V seguida de otra de +0.85 V y, por último, canal inactivo o sin transmisión, una señal de 0 V.

Para realizar cualquier conexión es necesaria un adaptador o tarjeta de red que cumpla los estándares adecuados de requisitos electrónicos.

En el nivel de enlace, se hace uso del protocolo MAC (Control de acceso al medio) que provee los servicios de asignación de dirección MAC, envió de datos, recepción de datos y controles de flujo. Utiliza solamente topología en bus o en estrella. También existen Fast Ethernet y Gigabit Ethernet, ofreciendo una mayor velocidad de transmisión como principal característica.

2.2.2 Token Ring [7]

Es conocida como el estándar IEEE 802.5. Desarrollada y comercializado por IBM, es junto a Ethernet una de las arquitecturas de red más conocidas actualmente. Utiliza una técnica de control de acceso al medio llamada técnica de **anillo con paso de testigo**. Dicha técnica, se basa en la transmisión de una pequeña trama llamada testigo o token, cuya principal funcionalidad es la de poder transmitir datos a la red cuando se disponga de dicho token o testigo. Cuando un dispositivo dispone de dicho token, podrá comenzar a transmitir los datos, mientras el resto de los dispositivos esperaran a la recepción del token. El transmisor

enviará el token, mediante la técnica *round-robin* [8], al próximo dispositivo, si ha terminado de transmitir o bien, ha empezado a recibir los datos que él está transmitiendo. A diferencia que Ethernet, solo puede transmitir un dispositivo a la vez en la red, que es el que dispone del testigo o token. Requiere una unidad central que hace de intermediaria llamada MSAU (Multistation Access Unit), en la que se realizará una conexión punto a punto con todos los dispositivos de la red. Una actualización de Token Ring en 1997 define una nueva técnica de control de acceso al medio llamada **anillo con paso de testigo dedicado**. En la que se configura el concentrador central MSAU como un conmutador de la capa 2 del modelo OSI.

2.2.3 CAN [9]

Utilizado principalmente en el automovilismo, actualmente se ha expandido su uso para aplicaciones de automatización industrial. Es un protocolo de comunicación serie para aplicaciones de tiempo real, que utiliza una topología de bus común para la comunicación con diferentes dispositivos. Dicho bus común requiere de dos resistencias en los extremos para el acoplamiento de los diferentes dispositivos. CAN opera en las capas inferiores del modelo OSI, la capa física y la capa de enlace de datos. En el nivel físico, CAN utiliza la codificación NRZ (No Return to Zero), la cual generalmente especifica un '1' lógico para una tensión positiva y un '0' lógico para una tensión negativa, sin que nunca llegue a ser 0 V. En el nivel de enlace de datos utiliza el protocolo MAC, al igual que Ethernet. Provee de 11 bits para direccionar los diferentes dispositivos conectados y mecanismos de detención de errores. CAN utiliza un par diferencial para la transmisión de datos que le proporciona una mayor robustez ante interferencias. La especificación de CAN se encuentra en el ISO 11898 para aplicaciones de alta velocidad y en el ISO 11519-2 para aplicaciones de baja velocidad.

La transmisión a alta velocidad puede llegar hasta velocidades superiores a 1 Mbit/s para aproximadamente 40 metros. Mientras que a baja velocidad puede alcanzar los 125 kbit/s. (A los 1000 metros su velocidad es de 50 kbit/s.) Las especificaciones CAN no especifican la estructura ni el diseño del propio bus físico. Es responsabilidad del dispositivo conectado al bus CAN, transmitir o detectar una de las dos señales llamadas dominante y recesiva. La señal dominante prevalece por delante de cualquier señal recesiva.

2.2.4 I^2C [10]

El bus I^2C o también bus de intercomunicación con circuitos integrados, es un bus de datos diseñado por Philips para la comunicación de circuitos integrados, comúnmente internos en los componentes de un ordenador. Los dispositivos se intercambian datos e instrucciones mediante dos líneas o cables bidireccionales. Una de las líneas se utiliza para la transmisión de datos e instrucciones SDA, y la otra para la sincronización de reloj SCL. Ambas líneas están alimentadas mediante una fuente de alimentación positiva, a través de un par de resistencias. Existen dos roles posibles: maestro y esclavo [11]. El maestro es el encargado de controlar las operaciones realizadas en el bus y de controlar a los esclavos. Los esclavos cumplen con las operaciones especificadas por el maestro. El maestro se encarga siempre de iniciar la transferencia de datos. Para iniciar una transferencia de datos es necesario que el bus este libre.

Utilizado generalmente para la comunicación interna de un circuito integrado. Su velocidad de transmisión oscila entre 0,1 Mbit/s y 3,4 Mbit/s en modo bidireccional y hasta 5Mbit/s en modo unidireccional.

2.2.5 RS-485 [12]

Este estándar, sólo define la capa física del modelo OSI. No define ningún protocolo para la conexión entre los diferentes dispositivos. RS-485 se define como una interfaz eléctrica y no un protocolo. Utiliza una topología de bus lineal, en el que no se pueden superar un número máximo de 32 dispositivos. Utiliza una transmisión diferencial de datos en serie, lo que significa que mide la diferencia de tensión entre dos señales. La conexión se realiza a través de cable de cobre de par trenzado, siendo esta capaz de transmitir en uno u otro sentido, pero no de manera simultánea (half-duplex). Posee una única alimentación de 5V. Es resistente al ruido exterior gracias a la transmisión diferencial. Su longitud máxima de 1200 metros a una velocidad de 100 kbit/s. Y con una velocidad máxima de transmisión de 10 Mb/s a 12 metros. Solo se recomienda para conexiones punto a punto, descartando el resto de las topologías. Es capaz de funcionar en técnicas maestro esclavo, cambiando su manera de trasmitir y recibir datos, siendo ahora posible realizarlo de manera simultánea (full-duplex) en este último caso utilizando 4 cables en lugar de 2.

2.2.6 HomePlug (PLC) [13]

Actualmente el nombre HomePlug, hace referencia a varios estándares que presentan una alternativa a la red doméstica, haciendo uso de la red eléctrica. El primer estándar fue HomePlug 1.0, seguido posteriormente de HomePlug AV (Audio y Video) y HomePlug AV2, que finalmente pasaría a ser el estándar IEEE 1901.

El estándar especifica dos capas físicas, una basada en la Transformada de Fourier y la otra basada en Wavelet (transformadas matemáticas), ambas mediante multiplexación por división de frecuencia ortogonal (OFDM). Para el control de acceso al medio (MAC) y por consiguiente la capa de enlace de datos del modelo OSI, se especifican también dos capas, ambas necesarias, la capa de redes domésticas y la capa de acceso a Internet.

Este estándar ha sido una opción para tener en cuenta para realizar nuestro proyecto, pero lo descartamos puesto que la implementación de este era compleja y los módulos que debíamos adquirir para realizar la comunicación eran más caros. Este estándar ofrece una mayor velocidad de transmisión que nuestro protocolo propio, pero esto no es una característica sustancial para tener en cuenta en nuestro proyecto, ya que es prioritario un módulo más sencillo y de menor coste.

2.2.7 ARINC [14]

Utilizado en la aviación para la transmisión de datos. La especificación ARINC 429 es la más utilizada. Utiliza un par de cables de par trenzado para la transmisión de datos, mediante una interfaz eléctrica bipolar. Dicho par de cables lleva una señal diferencial equilibrada, con una diferencia de tensión de 20V (+10V, -10V). Para la codificación de datos utiliza una onda de transmisión de datos de retorno a cero bipolares diferenciales. (BPRZ). Un solo par de cables limita a menos de 20 receptores. El transmisor envía datos constantemente. El formato de la palabra se limita a 32 bits.

2.2.8 RS-232 [15]

Es un estándar del puerto serie que designa una norma, para la transmisión y recepción de datos por medio del puerto serie. Ofrece una velocidad de transmisión muy baja, pero suficiente para enviar poca cantidad de datos. Usaremos esta interfaz para realizar nuestra conexión mediante la línea eléctrica, dada su sencillez y fácil gestión, nos servirá para realizar nuestras pruebas. Este estándar ha sido prácticamente reemplazado por la interfaz

USB, más rápida y fiable, aunque aún se sigue utilizando el puerto serie. Por último, destacamos que la transmisión de datos se realiza de manera asíncrona, utilizando tan sólo dos cables, una señal de transmisión Tx y una señal de recepción Rx.

2.2.9 SPI [16]

Estándar de comunicaciones (Serial Peripheral Interface), utiliza la técnica maestro-esclavo al igual que la tecnología I^2C para la comunicación entre dispositivos. Se necesitan una línea de reloj, una entrada de datos, una salida de datos y un selector de esclavo, para indicar el esclavo con el que se desea comunicar. La comunicación es en ambos sentidos (Full-duplex), lo que significa que ambos maestro y esclavo podrán enviar y recibir datos simultáneamente, aunque solo puede existir un maestro en el sistema SPI. Los esclavos se pueden conectar en cascada para ampliar los registros de un esclavo.

2.2.10 FDDI [17]

Conocida como la interfaz de datos distribuida por fibra, consta de una topología de red en anillo haciendo uso de la arquitectura de red Token Ring. Opera en la capa física y capa de enlace de datos del modelo OSI. Además, define una estructura de subcapas básicas. PMD o dependencia del medio físico, PHY o protocolo de la capa física, MAC o control de acceso al medio y SMT o gestión de estaciones.

2.3 Topologías de red [18]

Dentro de las topologías de red podemos distinguir dos clasificaciones más comunes:

- Topología física, distribución espacial en la que se encuentran los dispositivos formando la red.
- Topología lógica, es la distribución que sigue el flujo de datos en una red.

2.3.1 Topología física

Nos centramos en cómo se distribuyen los elementos físicamente, es decir, su disposición en el espacio.

2.3.1.1 Topología punto a punto

Es la topología más simple y la base del resto de topologías. Un enlace de un punto de origen a otro punto destino para la comunicación entre ambos. Ofrece una conexión permanente y sin impedimentos con otro dispositivo.

2.3.1.2 Topología de bus común

Se compone de una dorsal principal sobre la que se conectan los diferentes dispositivos para comunicarlos entre sí. Dicha topología es usada en el protocolo Ethernet, CAN e I^2C. Generalmente es sencillo añadir y quitar dispositivos que se encuentren en dicha topología de red. Un fallo en un dispositivo dejará intacta la red, mientras que un fallo en el bus provocará un fallo en toda la red.

2.3.1.3 Topología en estrella

Todos los dispositivos están conectados a un nodo central por el que pasan todos los paquetes de la red. Este nodo central reenvía toda la información recibida del resto de nodos o dispositivos incluso al mismo que lo envío. Los nodos centrales pueden ser switches o hubs. Si falla cualquier nodo o dispositivo conectado al nodo central, simplemente se aislará dicho

nodo o dispositivo. En cambio, si falla el nodo central, fallara toda la red, dando lugar a un punto crítico de fallo. Esta topología es utilizada por la tecnología Token Ring.
Cuando se requieren de varios nodos centrales, se crea una nueva variante conocida como topología en **estrella extendida**. La topología en estrella extendida es la que se utiliza en las redes telefónicas.

2.3.1.4 Topología en árbol

Llamada también topología jerárquica, en la que existe un nodo raíz que distribuye la carga con otros nodos de nivel inferior. En esta topología los nodos hoja son los que finalizan la jerarquía en niveles. Parecida a la topología en estrella extendida, pero no cuenta con un nodo central, sino que existe un nodo troncal sobre la que se ramifican otros nodos. Si un nodo intermedio falla, quedarán aislados todos los nodos pertenecientes a esa ruta. En caso de fallar el nodo raíz, al igual que la topología en estrella, toda la red quedara aislada, siendo un punto crítico de fallo.

2.3.1.5 Topología en anillo

Todos los dispositivos están conectados mediante una entrada y una salida a diferentes nodos, formando una disposición en círculo. Todos los nodos tienen un transmisor y un receptor, de manera que de comunican entre sí mediante un token o testigo, para indicar el turno de transmisión de cada nodo. De esta manera, se evitan colisiones en la red.

Existe además una topología llamada **anillo doble**, en la que cada nodo está conectado con el nodo anterior y con el siguiente mediante una conexión doble, en lugar de una única conexión como sucede en una topología en anillo.

2.3.1.6 Topología en malla

Distinguimos dos variaciones:

- Malla parcial, un nodo en la red está conectado con algunos nodos de la red.
- Malla completa, un nodo en la red está conectado con todos los nodos de la red.

Es la topología más robusta. En la disposición de malla completa, no existen interrupciones ni colisiones en la red, ya que todos los nodos tienen una conexión punto a punto con el resto de los nodos. En caso de que se produzca un fallo en uno de los nodos, simplemente aislará a ese nodo y en el caso de que se produzca un fallo en la conexión punto a punto, se tomará otro camino para la comunicación con ese nodo. Por contraposición, la conexión punto a punto con todos los nodos resulta muy costosa y a medida que se añaden más nodos a la red, se requerirán de un mayor número de conexiones. Por lo que pese a ser muy rápida y confiable, su coste es muy elevado.

2.3.1.7 Topología híbrida

Combinan dos o más de las topologías de las anteriores para adaptarlas a las diferentes situaciones que nos podamos encontrar. Es muy compleja de administrar y mantener puesto que diferentes topologías requieren diferentes segmentos de comunicación.

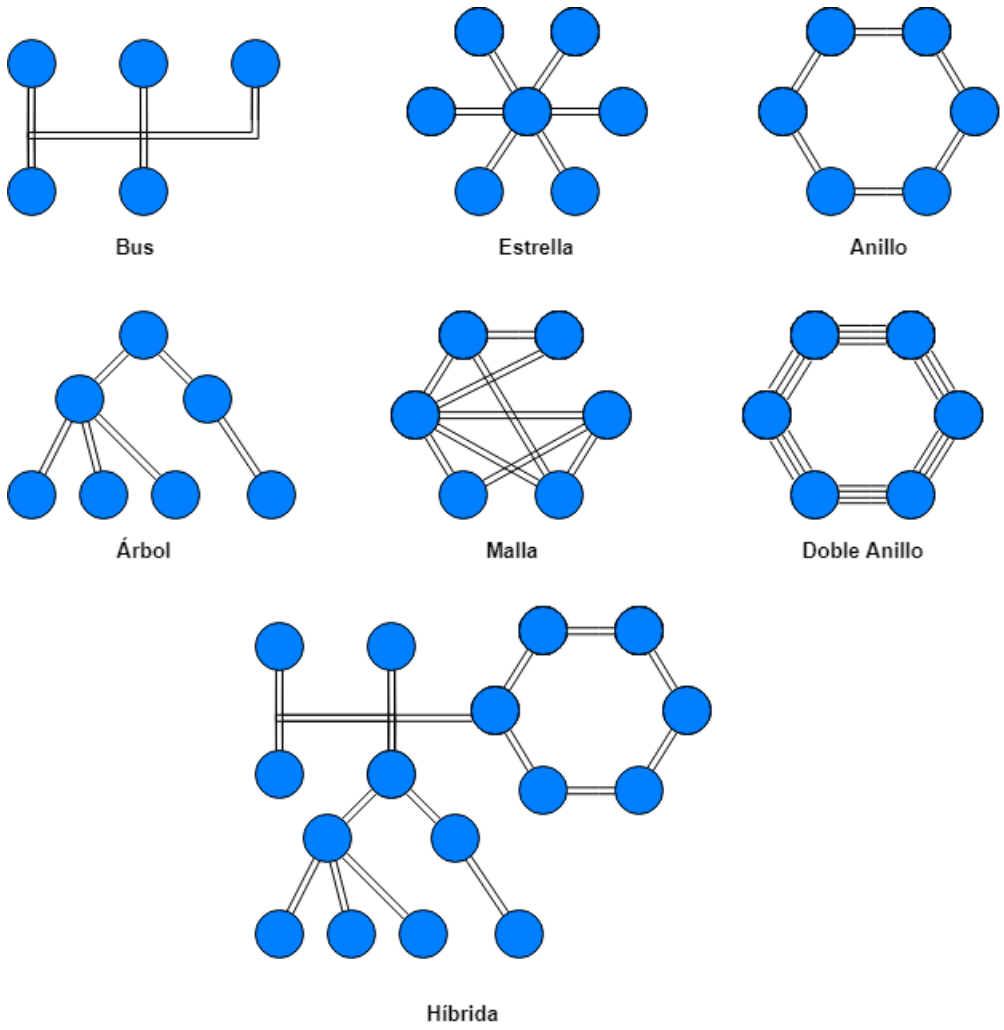

Figura 2.1. Topologías de red.

2.3.2 Topología lógica

Nos centramos ahora en cómo se realiza el flujo de datos, sin importarnos como sea la distribución espacial de los dispositivos.

2.3.2.1 Topología en anillo

Conocida como topología activa, los datos fluyen de un dispositivo a otro hasta que lleguen al dispositivo destino. Cada dispositivo espera su turno para poder regenerar la señal y transmitir los datos. Esta topología es usada por la tecnología Token Ring.

2.3.2.2 Topología de bus

Conocida como la topología pasiva, no regenera la señal. Los datos se transmiten por el bus hasta el dispositivo destino. Es posible que se necesiten unos dispositivos especiales de red llamados repetidores, que se encargan de regenerar la señal para largas distancias de transmisión.

3 Diseño

La elección en el diseño ha sido libre, pudiendo escoger entre múltiples alternativas. Hemos intentado que nuestra elección sea lo más próximo a un escenario real, en el que deberemos lidiar con diferentes elementos muy diferentes entre sí. De esta manera, podremos trabajar en un contexto más enfocado a los problemas reales.

Para conseguir una buena compresión sobre las limitaciones de nuestro proyecto, es necesario conocer cada uno de los componentes que lo integran. Es por ello, que se facilitan varios esquemas visuales para ayudar a comprenderlo de manera sencilla.

3.1 Marco de trabajo

Uno de los mayores sectores de crecimiento en tecnologías y servicios de comunicaciones es el área de IoT, que pretende conectar todos los dispositivos a la gran red de redes de Internet, dotándoles a estos de una parte de inteligencia.

Nuestro entorno de trabajo en este proyecto es limitado, dado que es un área muy extensa, no podríamos abarcarlo todo en este proyecto.

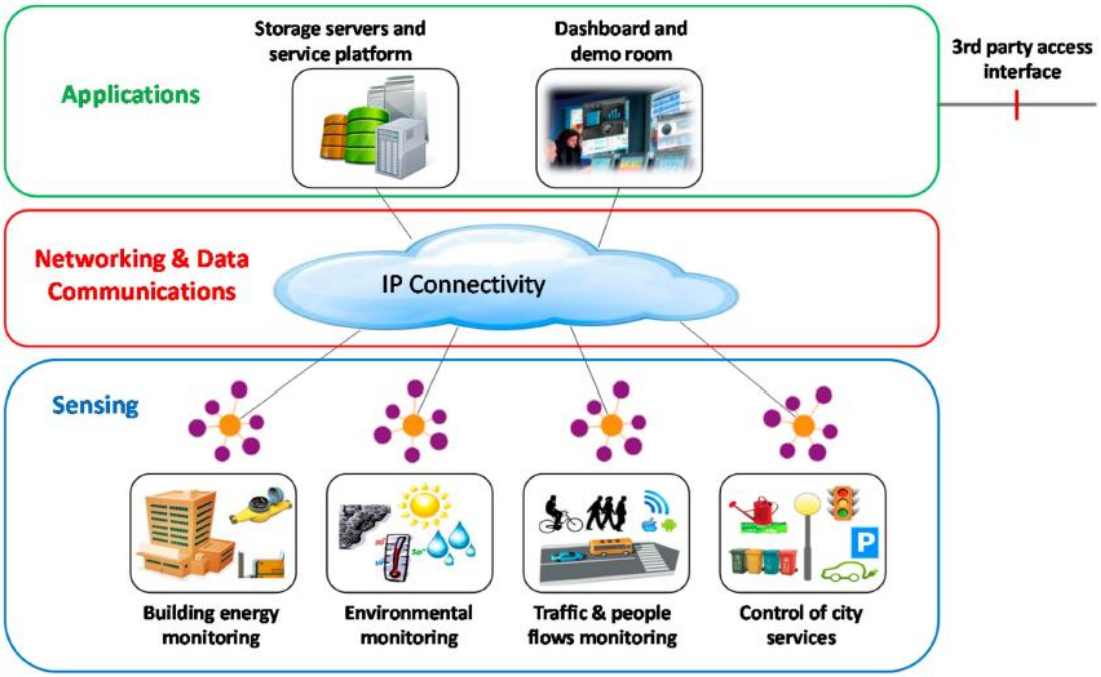

Figura 3.1. Área de trabajo del proyecto. **[19]**

En la **Figura 3.1**, podemos observar una generalización de una distribución de capas de los diferentes módulos de los que se compone una red en el área de IoT. Nuestro proyecto solo desarrolla la capa más baja, la de Sensado (Sensing). El resto de las capas son necesarias, pero no se abordarán en este proyecto, de manera que servirá de base para futuros proyectos.

En la capa de Sensado, se realizan las conexiones a nivel físico y de enlace referentes en el modelo OSI de las capas de red. Tendremos comunicación entre dispositivos de una red local, pero no se llega a realizar comunicación alguna a través de Internet, por lo que el

proyecto solo se centrará en la comunicación a nivel de red local. Sin embargo, poseemos un servidor web, capaz de recibir datos a través de Internet, pero este soló se ha realizado con fines académicos y para pruebas, por lo que no posee métodos de seguridad.

3.2 Esquema del experimento (conceptual)

Hemos realizado nuestro proyecto basándonos en un esquema bastante sencillo dividido en niveles. Cada nivel engloba una parte diferenciada que pasamos a comentar ahora:

- Nivel 0, está compuesto por solo un elemento, que procesa las peticiones que provienen de Internet, aunque en nuestro caso solo se realizará a nivel local y no a través de Internet, enviando las peticiones a los dispositivos de nivel 1 a los que vaya dirigida la petición. La conexión entre el dispositivo de este nivel con Internet no es relevante para este proyecto, puesto que se está operando a nivel local y nunca a nivel de red. La conexión entre el dispositivo de este nivel y las de nivel 1, se realizará mediante la línea eléctrica. (PLC)
- Nivel 1, posee varios dispositivos. Cada uno de estos dispositivos, realizará una conexión que podrá ser de cualquier tipo, dado que queremos tener alta escalabilidad en nuestro proyecto. Nosotros elegiremos I^2C como medio de comunicación con los elementos sensores o actuadores.
- Nivel 2, está constituido por los elementos sensores o actuadores de los que deseemos recoger datos, o realizar algún tipo de acción.

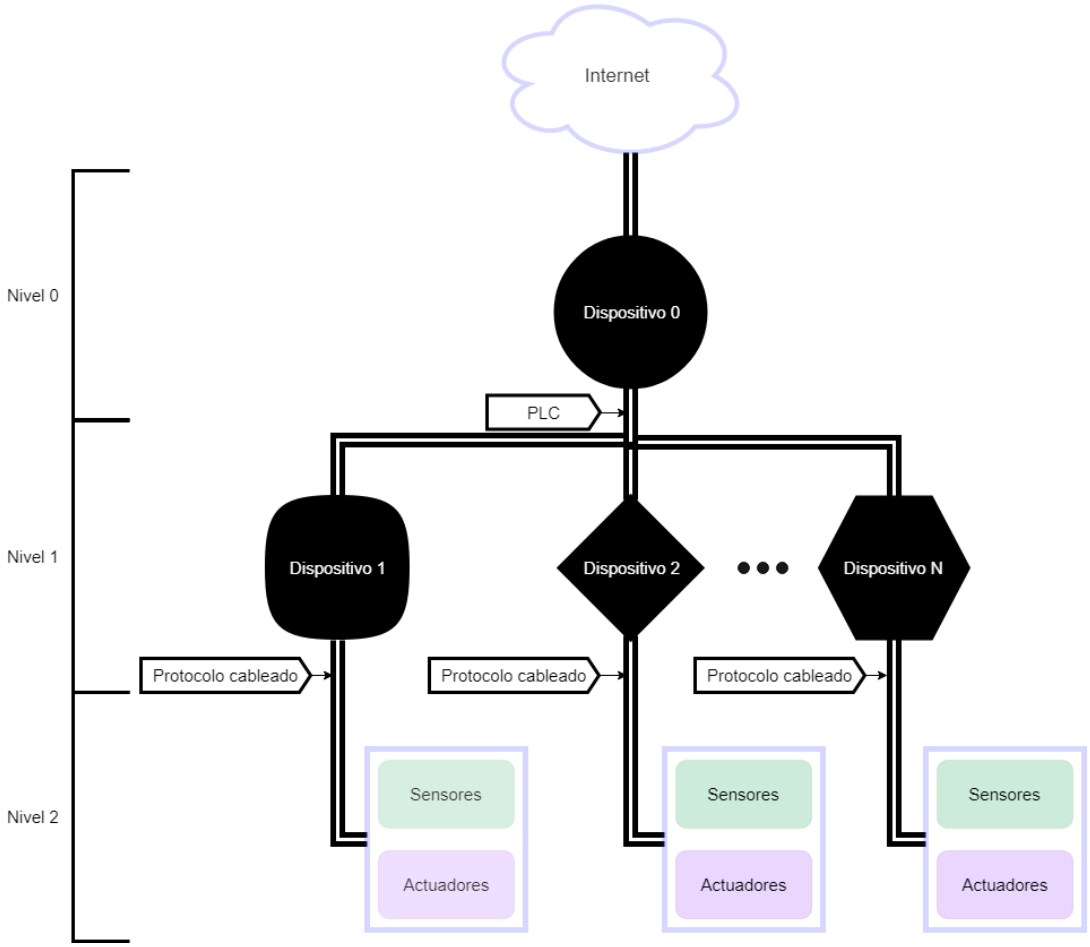

Figura 3.2. Esquema completo del proyecto.

Como podemos observar en la Figura 3.2, nuestro proyecto sigue una topología física híbrida, dado que combina diferentes métodos de comunicación y una topología lógica en bus, dado que todas las peticiones se agrupan en un único punto.

Los dispositivos son sistemas empotrados, tales como una placa Raspberry Pi [20], Arduino [21] o BeagleBone [22].

La configuración de dichos elementos sigue un modelo llamado maestro-esclavo. La técnica maestro-esclavo, se basa en que el maestro solicita ciertos requerimientos y el esclavo sirve dichos requerimientos, respondiendo a su solicitud. Los dispositivos pertenecientes a los esclavos poseen elementos sensores y actuadores tales como un sensor infrarrojo, varios leds y una pantalla. Los esclavos, envían y recogen datos a dichos elementos, para que estos modifiquen su estado, o simplemente para recoger datos en un momento determinado.

3.3 Funciones que implementar

Para nuestro experimento, vamos a desarrollar cierta funcionalidad para probar de manera sencilla nuestro experimento.

La funcionalidad para desarrollar será la siguiente:

- Función 1: Apagar y encender leds, teniendo 16 leds, indicar número de led a accionar.
- Función 2: Reiniciar un contador mostrado en una pantalla oled.
- Función 3: Obtener datos de un sensor infrarrojo.

3.4 Software

En esta sección, cubriremos todos los aspectos referentes a elementos software utilizados en nuestro proyecto. También abarcaremos los lenguajes de programación necesarios para la realización de toda la funcionalidad, explicando su utilidad.

3.4.1 Lenguajes de programación

En esta sección abordaremos todos los lenguajes de programación utilizados a lo largo de todo nuestro proyecto. Debemos destacar, que la elección de dichos lenguajes ha sido personal, ya que se podría haber escogido cualquier otro lenguaje de programación, como por ejemplo C, pero se han escogido estos por estar familiarizado con ellos entro otros aspectos.

3.4.1.1 Python [23]

El lenguaje principal de nuestro proyecto es Python. Toda la funcionalidad por implementar y la comunicación se ha realizado a través de este lenguaje. Este lenguaje ha sido elegido por ser uno de los referentes actualmente y porque está dotado de gran potencial, funcionalidad y documentación.

Para la simplicidad de nuestras tareas, y con la idea de no reinventar la rueda, hemos hecho uso de múltiples librerías de código que ofrece este lenguaje. A continuación, se listan todas ellas y se detalla de manera generalista su principal funcionalidad en este proyecto:

- Adafruit_Python_SSD1306, utilizada para controlar la pantalla OLED.
- smbus2, utilizada para realizar de manera más cómoda la conexión I^2C.
- Adafruit_CircuitPython_VL53L0X, utilizada para el control del sensor infrarrojo.

- pySerial, utilizada para realizar la comunicación serie mediante línea eléctrica (PLC).

3.4.1.2 HTML [24], CSS [25], PHP [26] y jQuery [27]

Esta combinación de lenguajes de programación ha sido usada en las últimas fases de nuestro desarrollo, para la realización de una pequeña página web, para tener un acceso más simple y visual de toda la funcionalidad de nuestro experimento. Dicha página web se encuentra en el dispositivo de nivel 0, en la cual, se desplegará un servidor http, en nuestro caso Apache 2 [28], que permitirá la comunicación de PHP con nuestro script de comunicación realizado en Python.

Hemos escogido HTML, para realizar la estructura de nuestra página web, CSS para el diseño de dicha estructura, PHP para realizar la comunicación con el servidor de Python y JavaScript y jQuery, para ofrecer funcionalidad instantánea al usuario.

3.4.2 Sistema operativo [29]

El sistema operativo utilizado es el propio de Raspberry, llamado **Raspbian**. Es un sistema operativo pasado en Unix, similar a Linux. Nos ofrece una interfaz de usuario completa, así como múltiples herramientas software para ayudarnos de manera más simple a realizar diferentes tareas. Al igual que nos ocurre con los lenguajes de programación utilizados para desarrollar nuestro proyecto, el sistema operativo escogido podría haber sido cualquier otro, pero puesto que hemos escogido una Raspberry Pi como dispositivo encargado de realizar todas las acciones, era el más cómodo y sencillo.

4 Desarrollo

Nuestro desarrollo, nos ha permitido ir realizando la implementación de manera gradual, de tal forma que comenzamos desarrollando funcionalidades simples, para posteriormente conseguir implementaciones más complejas. Por lo tanto, hacemos referencia al término *"divide y vencerás"* solucionando un problema complejo en formas más simples.

Hemos decidido presentar el desarrollo en una distribución por fases, las cuales hemos ido realizando de manera progresiva hasta llegar al final del proyecto. Dichas fases, componen el desglose de cómo ha ido desarrollándose todo nuestro proyecto, es decir, cada fase representa el trabajo que se ha ido realizando a lo largo del tiempo, hasta llegar al producto final. A continuación, mostramos el esquema más específico al desarrollo:

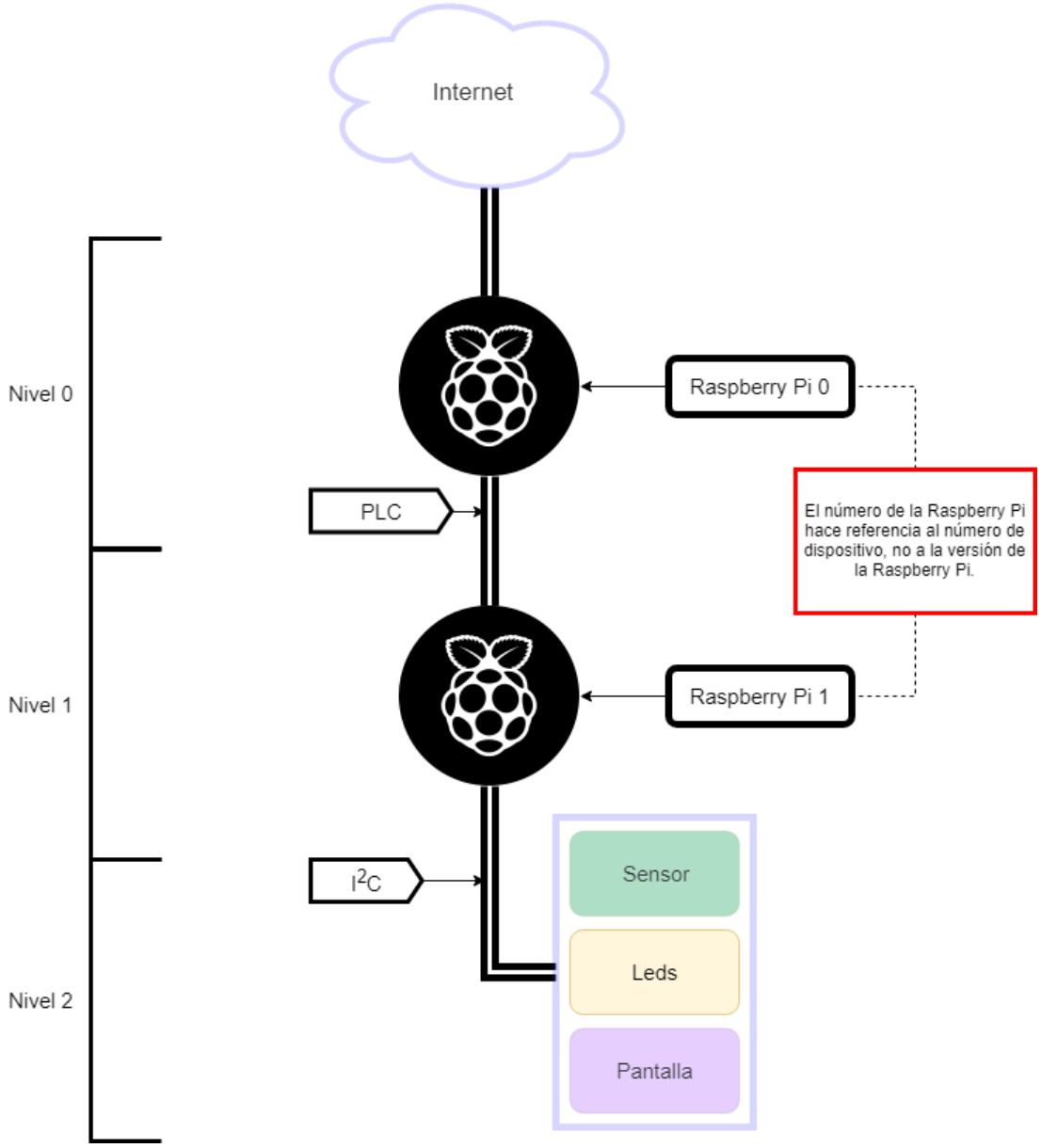

Figura 4.1. Esquema específico al desarrollo de nuestro proyecto.

En la **Figura 4.1**, podemos observar que nuestro esquema referente a la **Figura 3.1** se ha modificado, cambiando múltiples elementos por elementos hardware específicos, en concreto:

- **Dispositivos**, procederíamos a sustituir los dispositivos por las placas Raspberry Pi dado su buena relación de calidad y precio y sus buenas prestaciones. Aunque podríamos haber escogido otras placas como Arduino o BeagleBone.
- **Elementos sensores y actuadores**, encargados de recoger datos y realizar acciones en base a nuestra funcionalidad definida:
    - Sensor, escogeríamos un sensor infrarrojo llamado VL53L0X, capaz de recoger datos de proximidad del exterior.
    - Módulo de Leds, compuesto por el chip PCA9555 y 16 leds conectados a cada uno de sus pines.
    - Pantalla oled, hemos escogido la pantalla oled 0.91 por su bajo coste para el proyecto.
- **Protocolo I^2C**, el cual utilizaremos para conectar los elementos sensores y actuadores, con los dispositivos de nivel 1 de nuestro esquema.
- **PLC**, lo utilizaremos para realizar la conexión de los dispositivos de nivel 1 con el dispositivo de nivel 0, pero esta será a través de la interfaz RS-232 del puerto serie.

4.1 Primera fase

En esta fase, realizaremos las conexiones necesarias para llevar a cabo la comunicación I^2C referente a la **Figura 4.1**. Finalmente, llevaremos a cabo la implementación software, así como las pruebas necesarias, de los elementos sensores y actuadores que intervienen en nuestro proyecto.

4.1.1 Estructura

Definimos la estructura de los programas software sobre los que realizaremos la implementación de código. Tendremos pues un script de Python para cada elemento sensor, ya que así podremos tener la funcionalidad dividida y realizar pruebas en cada elemento, sin implicar a ningún otro. Así pues, tendremos los siguientes scripts:

- leds.py
- sensor.py
- pantalla.py

4.1.2 Implementación software

Para una mayor sencillez de implementación, recurrimos a hacer uso de las librerías que nos ofrece el lenguaje Python. A continuación, se exponen dichas librerías:

- Adafruit_Python_SSD1306, utilizada para controlar la pantalla OLED.
- smbus2, utilizada para realizar de manera más cómoda la conexión I^2C.
- Adafruit_CircuitPython_VL53L0X, utilizada para el control del sensor infrarrojo.

4.1.3 Conexiones

Para el correcto conexionado de dichos elementos, se expone a continuación, una muestra:

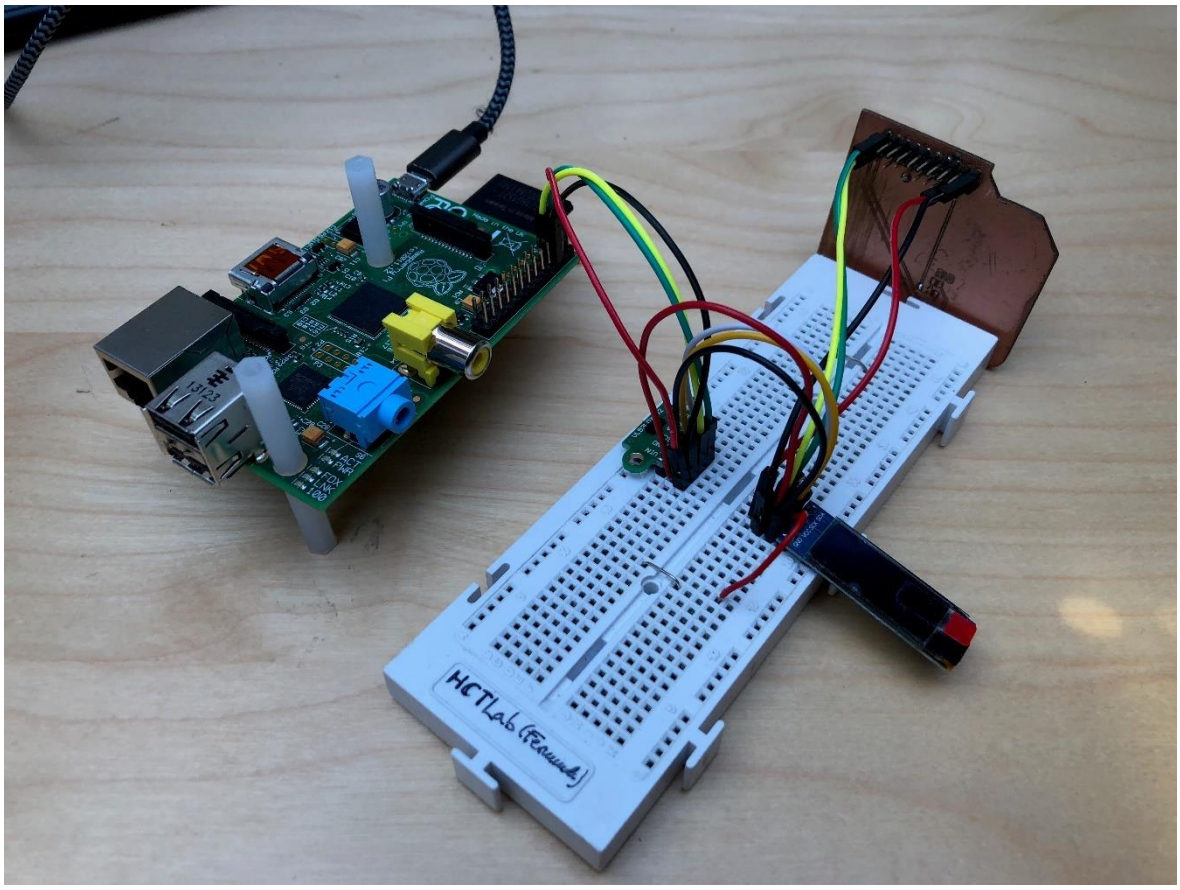

Figura 4.2. Conexionado de todos los elementos sensores y actuadores mediante I^2C.

La **Figura 4.2**, describe brevemente el cableado necesario para conectar los tres elementos sensores y actuadores de los que disponemos a una Raspberry Pi. La comunicación I^2C se lleva a cabo mediante dos cables, SDA, por el que se envían los datos y SCL para hacer la sincronización del reloj entre los dispositivos conectados. También podemos observar en la **Figura 4.2**, que nos hemos apoyado en el uso de una protoboard, para poder extender el conexionado de cables. Nuestro dispositivo, la Raspberry Pi, utiliza los pines GPIO 3 y 5, para la comunicación mediante I^2C. Adicionalmente, se debe llevar también la toma de tierra o GND y la alimentación de 3'3V a los elementos sensores y actuadores. Completando así de manera satisfactoria el cableado de la red I^2C.

4.2 Segunda fase

Continuamos nuestro desarrollo, realizando la conexión de varias Raspberry Pi mediante el puerto serie utilizando la interfaz RS-232, que será en un futuro adaptada para utilizar nuestro módulo de PLC. Esta es una fase muy importante, ya que, nos será de gran utilidad como medio de prueba para cuando posteriormente realicemos la conexión a través de la línea eléctrica conectando unos módulos de PLC.

4.2.1 Estructura

Hemos implementado una estructura de los programas llamada maestro-esclavo. Esta técnica hará que el programa maestro reciba las acciones a realizar, para que posteriormente se las envíe al esclavo y este ejecute dichas acciones, enviando una respuesta satisfactoria en caso de éxito o un error en caso contrario. A continuación, se describen más detalladamente las funciones que realiza cada componente.

4.2.1.1 Programa maestro

El programa maestro se encarga de recoger los comandos introducidos en la línea de comandos por el usuario y transmitirlos por el puerto serie para que este realice alguna acción con los elementos sensores y actuadores. También esperará una respuesta por parte del programa esclavo, dado que también deberá recibir los datos obtenidos de un sensor. Este programa lo hemos llamado **terminal.py**.

4.2.1.2 Programa esclavo

El programa esclavo se encargará de escuchar a través del puerto serie, las instrucciones que reciba por parte del programa maestro y de devolverle un mensaje satisfactorio. Dichas instrucciones se recogen en el manual de usuario, que no son más que una implementación de la funcionalidad definida en el apartado 3 de este proyecto. El programa esclavo lo hemos llamado **programa.py**.

4.2.2 Configuración del puerto serie en Raspberry Pi

Para probar el correcto funcionamiento del puerto serie, debemos realizar la configuración adecuada en nuestras Raspberry Pi, para poder utilizar este puerto para comunicación. En el **Anexo 1**, se detalla cómo hacerlo.

4.2.2.1 Interfaz del puerto serie en Raspberry Pi

El puerto serie en la Raspberry Pi, se encuentra físicamente en los pines GPIO 8 y 10 para todos los modelos de Raspberry Pi. El conexionado se explicará más adelante, sin embargo, cabe destacar que posee un pin de transmisión de datos y otro pin para el envío. En el sistema de Raspbian, la interfaz del puerto serie se encuentra en la ruta:

- /dev/ttyAMA0, para las versiones 1 y 2 de Raspberry Pi.
- /dev/ttyS0, para la versión 3 de Raspberry Pi.

Esta última ruta, se puede modificar a través del fichero de configuración de la Raspberry Pi, llamado **config.txt**.

4.2.2.2 Problemas de configuración Raspberry Pi 3

La Raspberry Pi 3, tiene un importante cambio al resto de generaciones predecesoras. Su puerto serie, está en otra interfaz diferente a la utilizada en el resto de Raspberry Pi por defecto. Para cambiarla, nos hemos visto en la obligación de documentarnos adecuadamente acerca de ello. Para una configuración adecuada del puerto serie de la Raspberry Pi 3 según se muestra en la documentación oficial de Raspberry, deberemos deshabilitar la interfaz bluetooth con el siguiente **comando** por terminal:

- sudo systemctl disable hciuart

Seguidamente, deberemos de añadir al fichero de configuración de la Raspberry Pi, **config.txt**, la siguiente línea de texto:

- dtoverlay=pi3-disable-bt

Una vez realizada la totalidad de la configuración, la Raspberry Pi 3 debería ser capaz de poder transmitir y recibir datos utilizando el puerto serie. En nuestro caso, la Raspberry Pi

era incapaz de comunicarse con otra Raspberry Pi, por lo que tuvimos que sustituirla por la Raspberry Pi 2.

4.2.3 Implementación software

Hemos realizado la implementación haciendo uso nuevamente de una librería de Python, llamada **pySerial**. Esta librería, nos proporciona una gran funcionalidad para el puerto serie, ayudándonos a abstraernos de un nivel más bajo de programación.

A nivel de código hemos realizado, como mencionábamos anteriormente, dos programas completamente distintos. Inicialmente, deberemos de realizar una pequeña prueba comentada en el **apartado 5**. Apoyándonos en la librería pySerial, procedemos a utilizar el objeto **Serial**, que nos permitirá abrir y cerrar el puerto serie, así como, enviar y recibir datos de este, con los parámetros deseados. La implementación de esta fase concluye con la comunicación adecuada entre ambas partes, pero sin tener en cuenta la implementación realizada en la **Primera Fase** de nuestro desarrollo, para poder llevar un control más gradual de nuestro proyecto.

4.2.4 Conexiones

La comunicación mediante el puerto serie se lleva a cabo utilizando dos cables, el cable TX, utilizado para la transmisión de datos y el cable RX, utilizado para la recepción de datos. Adicionalmente, deberemos conectar el cable de tierra (GND) entre los dos dispositivos, en nuestro caso, Raspberry Pi. Dichos cables se deberán cruzar entra ambas Raspberry Pi, para conseguir una comunicación adecuada.

En nuestro caso, disponemos de dos Raspberry Pi de modelos diferentes para realizar las conexiones, aunque esto no es un problema, dado que ambas utilizan los mismos pines GPIO 8 y 10 para la comunicación serie, se dejarán en el Anexo 1 la distribución de estos.

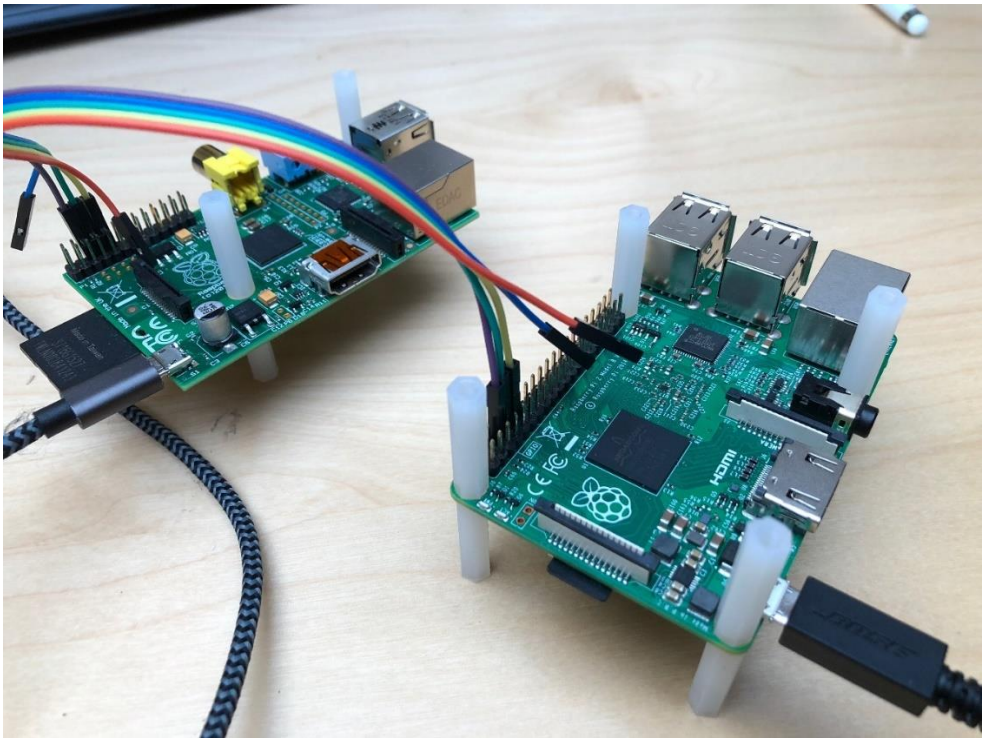

Figura 4.3. Conexionado mediante puerto serie de dos Raspberry Pi.

En la **Figura 4.3**, observamos el conexionado de dos Raspberry Pi utilizando el puerto serie para la comunicación y la línea a tierra. Como se comentaba anteriormente, deberemos cruzar los cables Tx y Rx entre las Raspberry Pi, para conseguir realizar la comunicación de manera adecuada.

4.3 Tercera fase

Nos centraremos en realizar la conexión de todos los componentes del proyecto, exceptuando los módulos PLC, así como, la implementación de un software capaz de acceder a los recursos del sistema de manera ordenada y simultánea. Para ello utilizaremos elementos propios de sistemas operativos, hilos [30] y semáforos [31].

4.3.1 Estructura

La estructura de esta fase ya empieza a parecerse a la estructura final de nuestro proyecto. Nuestro proyecto hasta ahora se componía de los siguientes componentes:

- terminal.py, programa maestro, encargado de recibir datos por terminal.
- programa.py, nuestro programa esclavo, sin poder aún realizar acciones sobre los elementos sensores y actuadores.
- leds.py, programa encargado de controlar los leds de un módulo de leds.
- pantalla.py, programa para que realiza el reinicio de un contador en una pantalla oled.
- sensor.py, encargado de recibir datos de un sensor y mostrarlos por terminal.

Esta estructura, se verá modificada de tal manera que toda la funcionalidad de los elementos sensores y actuadores, pase ahora a formar parte de nuestro programa esclavo, para que este realice acciones cuando recibe ciertos parámetros por terminal. Se expone seguidamente la nueva estructura del proyecto:

- maestro.py, de la misma manera, sigue siendo nuestro programa maestro que recogerá comandos por terminal para posteriormente enviárselos a nuestro programa esclavo, mediante la comunicación serie.
- esclavo.py, nuestro nuevo programa esclavo, se encargará de recibir los datos mediante la comunicación serie del programa maestro, así como de realizar el encendido y apagado de los leds, reiniciar el contador en la pantalla oled y de recibir los datos del sensor, mediante la comunicación I^2C, para enviárselos al programa maestro.

Esta nueva estructura, trae consigo numerosos cambios en la implementación de nuestro proyecto dado que, deberemos de hacer uso de semáforos e hilos para que nuestro programa esclavo pueda atender a todas las tareas que son requeridas.

4.3.2 Implementación software

Sufrimos múltiples cambios a lo que implementación se refiere. Nuevamente, y con la idea de no reinventar la rueda, volvemos a hacer uso de varias librerías de Python que nos ayudarán a realizar de manera sencilla el manejo de hilos y de semáforos, en este caso, la misma librería nos ofrece dos elementos clave necesarios, hilos y semáforos. Esta librería es:

- threading, de la que extraeremos tanto el manejo de hilos como de semáforos, dos elementos propios de sistemas operativos, indispensables en nuestro proyecto.

4.3.2.1 Hilos

En concreto de la librería de threading, utilizaremos el objeto **Thread** para crear los diferentes hilos de ejecución. En nuestro caso, hemos elegido que el proceso principal se mantenga escuchando al programa maestro por el puerto serie, mientras los hilos realizan el resto de las funcionalidades. Más detalladamente, crearemos **tres hilos**:

- Hilo 1, encargado de llevar a cabo el encendido y apagado de un led.
- Hilo 2, su función será reiniciar el contador de la pantalla oled.
- Hilo 3, cuyo cometido será el de recoger los datos de un sensor.

Estos hilos realizarán modificaciones sobre variables accesibles también por el proceso principal, que cuando lo necesite, consultará dichas variables para obtener datos o bien para realizar las acciones pertinentes. Cabe destacar que cada hilo, se quedará en un bucle infinito, realizando determinadas acciones según el papel del que les toque encargarse. Excepto el hilo encargado de la iluminación de los leds, ya que este sólo deberá realizar una acción determinada para morir posteriormente.

En el caso del hilo 1, este solo se ejecutará una vez y morirá, puesto que no debe realizar más acciones que esa, por lo que no se quedará esperando infinitamente. Por contrapartida, el hilo 2, se quedará realizando una actualización del contador cada segundo, para llegado el momento, reiniciar el contador. A continuación, el hilo 3 también permanecerá en un bucle infinito guardando los datos cada 5 segundos en una variable accesible por el programa principal para cuando sea necesario consultarlo. Finalmente, el proceso principal se encargará de todo el manejo de hilos y de la escucha activa por el puerto serie. Este proceso lanzará todos los hilos antes de comenzar la escucha activa por el puerto serie y se ejecutará de manera infinita. Existe una manera de parar la ejecución del programa esclavo, que se comentará en el manual del programador.

4.3.2.2 Semáforos

De la librería threading, utilizaremos también el objeto **Semaphore** necesario para crear semáforos y controlar así el acceso ordenado a mismas variables desde diferentes hilos de ejecución. Para nuestro proyecto, hemos definido tres semáforos:

- Semáforo 1, necesario para controlar el acceso del bus de datos de I^2C. Este semáforo, nos permitirá bloquear más de un acceso a este bus para que sólo haya un elemento que pueda acceder al bus y realizar lo que necesite.
- Semáforo 2, utilizado para bloquear el acceso simultaneo a la variable contador del módulo de la pantalla oled. En este caso, solo podrán acceder a dicha variable el proceso principal y el hilo correspondiente a la funcionalidad de gestión de la pantalla oled.
- Semáforo 3, su utilidad es bloquear el acceso a la variable global que recoge los datos del sensor, accesible exclusivamente por el hilo de ejecución encargado de la administración del sensor y por el proceso principal.

Finalmente, deberemos realizar las llamadas de los métodos acquire y release en las zonas de código críticas en las que necesitemos realizar el bloqueo para controlar el acceso a dichas variables mencionadas anteriormente o al bus I^2C.

4.3.3 Conexiones

En las dos fases anteriores, pudimos observar cómo realizar las conexiones pertinentes tanto para conectar los elementos sensores y actuadores a una Raspberry Pi, como las conexiones necesarias para conectar dos Raspberry Pi entre sí, mediante una comunicación a través del puerto serie.

Ahora, necesitaremos realizar en esta fase ambas dos, pero para el caso de los elementos sensores y actuadores, hemos hecho uso de un elemento adicional, llamado protoboard. La protoboard, es un elemento que nos ayuda a extender la señal. Gracias a ello, nos resultará más sencillo conectar el resto de los elementos sensores y actuadores a la Raspberry Pi. A continuación, se muestra una imagen con todas las conexiones:

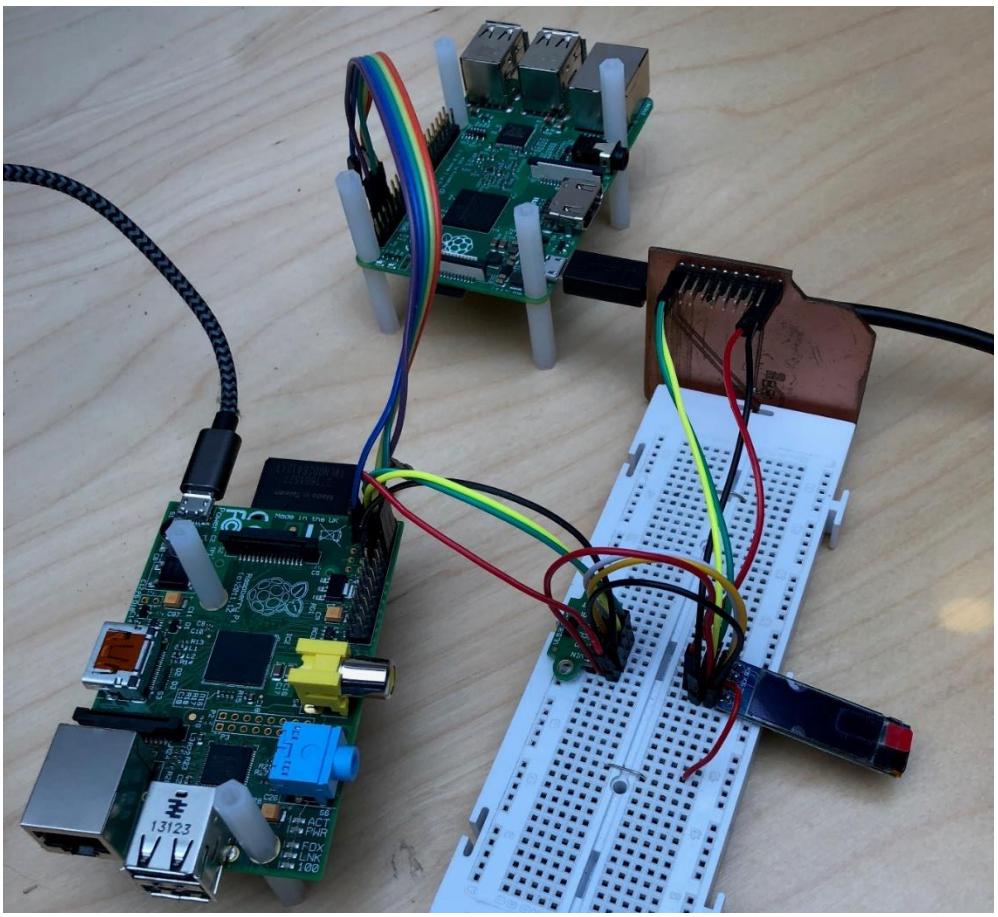

Figura 4.4. Conexionado de todos los elementos mediante el puerto I^2C y el puerto serie.

En la **Figura 4.4**, podemos observar que debemos extender las señales SDA y SCL de la Raspberry Pi, así como también las señales de 3’3V y GND, haciendo uso de la protoboard. Tras ello, iremos conectando cada uno de los elementos sensores y actuadores a dichas señales hasta tener a todos conectados simultáneamente con estas cuatro señales. Gracias a esta distribución mediante la protoboard, nos ha sido más fácil realizar el conexionado, en lugar de realizarlo sobre la Raspberry Pi.

4.4 Cuarta fase

Modificamos el programa maestro que mostraba un menú por la terminal de Linux, para implementar una página web, desarrollada en HTML5, PHP, CSS, Ajax y jQuery, que se apoyará en un script de Python para realizar las consultas al programa esclavo.

Para conseguir que nuestra Raspberry Pi sea capaz de recibir peticiones HTML, haremos uso de las facilidades que poseemos hoy en día y en lugar de crear nuestro propio servidor http, instalaremos un servidor Apache 2 en nuestra Raspberry Pi.

Como conclusión de esta fase, nuestro desarrollo constará de una página web, desde la que tendremos acceso de manera visual, a toda la funcionalidad implementada en las fases anteriores. Nuestro servidor http, se conectará mediante el script de Python al puerto Serie y enviará a la otra Raspberry Pi (programa esclavo) las acciones que deberá de realizar, para que esta las procese y envié lo necesario por el puerto I^2C, de manera ordenada.

4.4.1 Estructura

Esta fase difiere mucho con el resto de las fases, entre otros porque solamente nos centraríamos en mejorar la interfaz de usuario o como hemos descrito en fases anteriores, el programa maestro. Como explicamos en la fase 3, nuestra estructura del proyecto pasaría a tener exclusivamente dos componentes:

- maestro.py
- esclavo.py

Tras esta fase nuestra estructura sufrirá una gran modificación, pasando a ser mucho más extensa. A continuación, se detalla dicha estructura:

- **index.html**, estructura principal de la página web.
- **main.css**, hoja de estilos de la página web
- **main.py**, script que se comunica mediante el puerto serie con el programa esclavo.
- **main.js**, encargado de realizar las acciones requeridas por el usuario y enviárselas a main.php.
- **main.php**, encargado de recoger los comandos de la web y ejecutar el script main.py.
- **jQuery.min.js**, fichero externo a nuestra web, necesario para realizar consultas mediante jQuery.
- **TFG_uam_logo.png**, imagen con el logotipo de la UAM.
- **backend.py**, programa esclavo encargado de realizar las acciones necesarias enviadas por el programa maestro, será exactamente igual que esclavo.py.

Como podemos observar, la nueva estructura requiere de un servidor http para atender a las peticiones a través de la nueva página web, por lo que deberemos además configurar un servidor http en nuestra Raspberry Pi para dicho cometido. También observamos que no se han realizado modificaciones al programa esclavo, contenido en el script backend.py, que seguirá teniendo la misma funcionalidad que en la anterior fase, sin necesidad de adaptar nada. Finalmente, cabe destacar que esta funcionalidad se había definido como algo opcional a nuestro proyecto, puesto que nuestro principal cometido es el de realizar un módulo de comunicaciones, pero este diseño, supondría una interfaz de usuario mucho más visual y sencilla para realizar las acciones que lo que habíamos realizado anteriormente, haciendo uso de la terminal.

4.4.2 Configuración software

Esta fase al diferir mucho de las fases anteriores requiere una configuración algo especial para su correcto funcionamiento. Necesitaremos ejecutar en nuestra Raspberry Pi de nivel 0, un servidor http, capaz de procesar las peticiones que se realicen a través de la web. Podríamos realizar nuestro propio servidor http mediante Python, pero con tal de no reinventar la rueda, y puesto que, no era el objetivo principal de nuestro proyecto, haremos uso del servidor http Apache 2[REF].

La instalación de este servidor Apache 2, no es objetivo del proyecto, por lo que se dejará en el **manual de instalación**. Sin embargo, la configuración que deberemos realizar, si es interesante explicarla a continuación. El servidor Apache 2, sirve el contenido que se encuentra en la siguiente ruta:

- /var/www/html/

Por lo que deberemos de colocar nuestros ficheros en dicha ruta, en nuestra Raspberry Pi de nivel 0, a excepción del fichero backend.py, que se encontrará en las Raspberry Pi de nivel 1. Ahora, deberemos darle permisos de acceso a dicha ruta, para que pueda ser accesible por cualquier usuario, para ello ejecutamos el siguiente comando:

- sudo chmod 777 /var/www/html

De esta manera, permitiremos el acceso a los ficheros de cualquier usuario, y por lo tanto podrán ejecutar el script de Python. No obstante, aún deberemos darle permisos al puerto serie, para que el usuario pueda realizar la comunicación con el programa esclavo. Para ello, también deberemos dotar de permisos a la siguiente ruta:

- sudo chmod 777 /dev/ttyAMA0

Esta última acción es necesaria, ya que el puerto serie tiene limitaciones de acceso para evitar posibles ataques. Esto supondría un gran fallo de seguridad en nuestra arquitectura, pero no es tampoco objetivo de este proyecto, realizar una seguridad en nuestro sistema de comunicaciones.

Por último, para acceder debidamente a los contenidos web, deberemos acceder a la dirección IP de nuestra Raspberry Pi, en la que se encuentra escuchando en el puerto 80 en servidor Apache 2.

4.4.3 Implementación software

Como observamos anteriormente en esta fase, nuestro proyecto se compone ahora de múltiples ficheros adicionales a los de la anterior fase, por lo que se detallará a continuación, cada uno de ellos.

4.4.3.1 Index.html

Compone la estructura actual en HTML5 de nuestra página web. En dicha estructura tendremos 16 botones para controlar la iluminación independiente de cada led, un botón para el reinicio del contador y otro botón para la obtención de datos del sensor. Destacamos que no hemos utilizado un formulario como cabría esperar, ya que en su lugar haremos uso de

jQuery y Ajax, para la actualización de una parte de nuestra página, en lugar de actualizarla de manera completa.

4.4.3.2 Main.css

Es la hoja de estilos que modificará, según queramos diseñarla, nuestra página web para adaptarla visualmente a nuestras necesidades. Podemos destacar el uso de un elemento muy particular llamado slider, que nos ayudará a definir un botón de tipo encendido/apagado para nuestros leds.

4.4.3.3 Main.py

Este fichero es el encargado de realizar el envío del comando adecuado mediante la comunicación por el puerto serie al programa esclavo. No debe realizar comprobaciones, ya que en caso de ser necesarias deberían de estar en el programa esclavo. Es una pequeña adaptación del antiguo fichero terminal.py.

4.4.3.4 Main.php

Este fichero tiene múltiples cometidos importantes. Deberá recoger el comando accionado por el usuario, recogido por el fichero main.js, para posteriormente realizar la ejecución del fichero main.py con dicho comando pasado como parámetro. Por último, recogerá lo que devuelva el fichero main.py por terminal y lo devolverá al fichero main.js.

Para poder realizar la ejecución del fichero de Python main.py, teníamos varias opciones, nuestra elección fue la más sencilla y menos segura, haciendo uso del comando **exec** de PHP, por lo que se podrían generar brechas de seguridad en nuestro proyecto, que como comentábamos no es objetivo de nuestro proyecto. Otra posibilidad habría sido realizar la ejecución con un framework llamado Django [32]. Esto habría supuesto una mayor seguridad, pero también un mayor tiempo de desarrollo.

4.4.3.5 Main.js

Este fichero se encarga de recoger las acciones realizadas por el usuario en nuestra página web, de manera dinámica. Para ello hacemos uso de las consultas jQuery y Ajax, que nos permiten actualizar partes de nuestra página web, sin necesidad de recargarla por completo.

El resto de los ficheros restantes son externos a nuestra implementación y en el caso de jQuery.min.js, aportará funcionalidad adicional como si se tratase de una librería y en el caso de TFG_UAM_logo.png, únicamente aportará una imagen del logotipo de la UAM a nuestra web.

Finalmente, procedemos a mostrar la página web desarrollada a lo largo de esta fase:

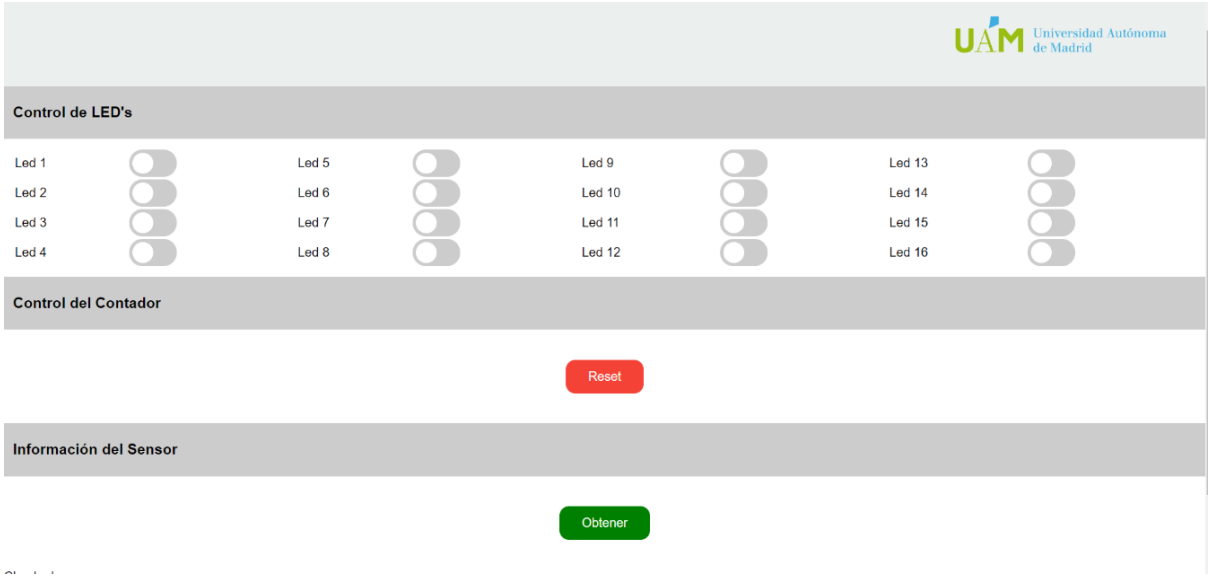

Figura 4.5. Muestra de la página web.

En la **Figura 4.5**, podemos observar, que disponemos de varios botones para encender o apagar un led determinado, reiniciar el contador de la pantalla OLED y un último botón, para obtener los datos del sensor.

4.5 Quinta fase

Esta es la última fase de nuestro desarrollo. En esta fase, realizaremos la conexión de ambas Raspberry Pi mediante los módulos KQ-330 utilizando la línea eléctrica para la transmisión y recepción de datos. Para esta fase se nos ha facilitado el osciloscopio, una herramienta que nos ayudará a realizar las pruebas necesarias para observar el funcionamiento del módulo KQ-330. También se nos facilita un diseño con conexión a línea eléctrica, para conectar dichos módulos y dos Raspberry Pi.

Además, añadiremos un nuevo fichero que se encargará de realizar la comunicación sobre la línea eléctrica, gestionando las diferentes peticiones de envío y recepción. Esta parte se explicará más detalladamente en el apartado 4.5.5 implementación software.

4.5.1 Estructura

Nuestra estructura, una vez más se verá modificada, aunque será un cambio mínimo. La estructura final añade un nuevo fichero para el control de manera simple de la interfaz por línea eléctrica, haciendo uso del puerto serie. A continuación, se muestra la estructura final del proyecto, que dividiremos en dos partes:

4.5.1.1 Nivel 0 - Raspberry Pi o Programa maestro

Esta parte será la correspondiente al nivel 0 mostrado en la **Figura 4.1**. Encargada de mostrar la interfaz de usuario, mediante el servidor http. La estructura, sufre algunos cambios mostrados en negrita, añadiendo un nuevo fichero para la gestión de envío y recepción de datos mediante línea eléctrica:

- index.html, estructura principal de la página web.
- main.css, hoja de estilos de la página web.
- **main.py**, script que se comunica con el programa esclavo mediante línea eléctrica.

- **plc.py**, módulo que contiene la clase SocketPLC, para realizar la conexión mediante línea eléctrica.
- main.js, encargado de realizar las acciones requeridas por el usuario y enviárselas a main.php.
- main.php, encargado de recoger los comandos de la web y ejecutar el script main.py.
- jQuery.min.js, externos a la implementación, ofrecen funcionalidad adicional al fichero main.js.
- TFG_UAM_logo.png, imagen externa a la implementación.

4.5.1.2 Nivel 1 - Raspberry Pi o Programa esclavo

Esta otra parte, hace referencia a las Raspberry Pi pertenecientes al nivel 1 expresado en la **Figura 4.1**. Esta parte es la encargada de realizar las acciones requeridas por el usuario mediante la línea eléctrica, comunicando dichas acciones mediante el puerto I^2C a los elementos sensores y actuadores. Los ficheros con cambios se expresan en negrita.

- **backend.py**, con la capacidad de controlar los elementos sensores y actuadores mediante la comunicación I^2C y realizar acciones enviadas por el usuario mediante la línea eléctrica.
- **plc.py**, módulo que contiene la clase SocketPLC para ayudarnos a realizar de manera sencilla la conexión mediante línea eléctrica.

4.5.2 Módulo KQ-330

Lo elementos clave en nuestro proyecto son los módulos KQ-330. Se proporciona una descripción más detallada sobre estos módulos en el Anexo 2. A grandes rasgos, estos módulos transforman nuestra señal para transportarla mediante la línea eléctrica y posteriormente volver a transformarla para que el receptor pueda comprender los datos. En la **Figura 4.6**, se presenta el mencionado módulo:

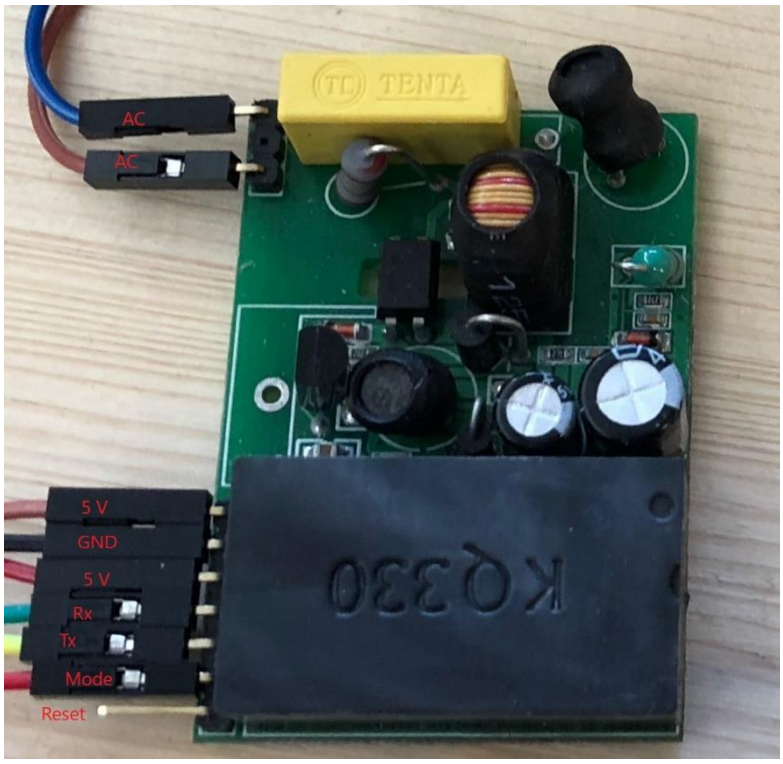

Figura 4.6. Conexiones en módulo KQ-330.

La hoja de datos de este módulo era incomprensible, por lo que hemos tenido que hacer uso de lo que se conoce como ingeniería inversa, para descifrar la mencionada hoja de datos. No obstante, el uso del osciloscopio ha sido indispensable para observar el comportamiento de estos módulos con diferentes configuraciones. Este proceso ha llevado algo de tiempo, pero finalmente hemos podido dar con la receta adecuada para la realizar la correcta configuración de estos módulos.

Los módulos KQ-330, poseen un protocolo simple para el envío y recepción de datos. Este protocolo se define como: el primer byte para fijar la longitud de los datos que queremos enviar y a continuación, los datos de envío. Mediante pruebas para comprobar la longitud máxima de datos de envío, pudimos observar que el tamaño máximo de los datos que podemos enviar es de 252 bytes. A continuación, se presenta la trama de línea eléctrica:

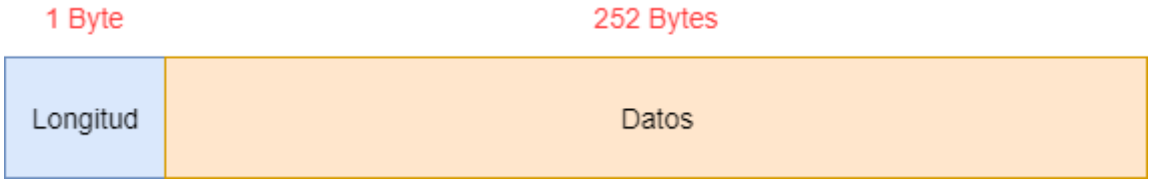

Figura 4.7. Trama de datos que gestionan los módulos KQ-330.

Como observamos en la **Figura 4.7**, este protocolo no se encarga de la gestión de múltiples dispositivos, por lo que para nuestro proyecto deberemos realizar una modificación de este protocolo, para añadir nueva funcionalidad mediante la línea eléctrica.

4.5.3 Protocolo propio

Como comentábamos al final del apartado anterior, necesitaremos definirnos un protocolo nuevo, capaz de poder gestionar múltiples dispositivos a la vez. Esta sección es **crítica y la más importante de todo el desarrollo**, ya que es en esta sección sobre la que se sostiene todo nuestro trabajo, por lo que, analizando diferentes opciones hemos decidido que el protocolo se defina como la figura mostrada a continuación:

Figura 4.8. Protocolo para gestión de múltiples dispositivos.

Como observamos en la **Figura 4.8**, aparecen nuevos campos en la trama con respecto a la **Figura 4.7**. Pasamos ahora a mencionar cada uno de ellos, dando una pequeña explicación:

- **Longitud**, es el tamaño que tendrá la trama que se desea enviar.
- **Origen**, es el identificador único del dispositivo que realizará el envío de datos.
- **Destino**, es el identificador único del dispositivo receptor de los datos.
- **Datos**, los datos que enviaremos al otro dispositivo.
- **CRC** [33], es el código de verificación de redundancia cíclica, que nos ayudará a comprobar si la trama no ha sufrido ninguna modificación.

Con este nuevo protocolo, añadimos los campos necesarios de origen y destino para poder realizar la comunicación con más de un dispositivo a través de la línea eléctrica, y además el campo CRC nos permite comprobar que estos datos han llegado correctamente sin sufrir ningún cambio. Consideramos pues, que estos campos son suficientes como propósito para nuestro proyecto, aunque puede ser modificados si se desea para una futura mejora.

4.5.4 Configuración hardware

En nuestro desarrollo, no contábamos con que el voltaje de los módulos KQ-330 fuese de 5V, por lo que hemos tenido que hacer una configuración algo más especial haciendo uso de los llamados **conversores de nivel**. Esto elementos hardware nos permitirán convertir las señales pertenecientes a la Raspberry Pi de 3’3V en señales de 5V, para poder utilizar los módulos KQ-330. Una vez estos módulos se comuniquen entre sí, volveremos a convertir la señal, esta vez de 5V a una señal de 3’3V, para no dañar así la Raspberry Pi.

Estos elementos son indispensables en nuestro proyecto, puesto que sin ellos nos arriesgaríamos a romper tanto los módulos KQ-330, como las Raspberry Pi.

4.5.5 Implementación software

Como ya adelantamos en el apartado 4.5.1, vamos a realizar la implementación de un nuevo fichero, llamado plc.py. Este fichero contiene una definición para la gestión del envío y recepción de datos mediante la línea eléctrica, que pasamos a detallar a continuación.

El fichero plc.py, contiene una nueva clase **SocketPLC** de creación propia, que realiza la configuración necesaria del puerto serie. Esta clase, recibe como parámetro el identificador, que deberá ser único, del dispositivo actual. Posee dos métodos, uno para realizar el envío de los datos, y otro para la recepción de datos. Estos dos métodos se han definido para seguir el protocolo propio, establecido en el apartado 4.5.3, de tal manera que en caso de que los datos que recibe un dispositivo no son para él, los descartará y seguirá a la espera de datos que sí sean para él.

También se ha implementado la funcionalidad de respuesta de un mensaje recibido, es decir, cuando se realice un envío de manera satisfactoria, el emisor se esperará un mensaje de respuesta por parte del receptor, para indicarle que todo ha ido bien. Si supera un umbral de 25 segundos, tiempo máximo de envío de datos, el emisor entenderá que no se ha podido recibir el mensaje correctamente. Adicionalmente, el envío cuenta con un reenvío de los datos en caso de que el receptor no le devuelva el mensaje adecuado.

4.5.6 Problemas software

Con la nueva incorporación de los módulos KQ-330, al realizar la transmisión de los datos por la línea eléctrica, estos tardan algo más de tiempo de lo que transcurría con el puerto serie. Concretamente, tras la realización de pruebas de comprobación para el envío de una trama máxima mediante la línea eléctrica, se detectó que el envío de una trama de 253 bytes tardaba en realizarse 24,8 segundos. Estos resultados no eran los que esperábamos, aunque son suficientes para este proyecto. Esto se debe a que los módulos KQ-330 realizan el envío de los datos byte a byte en cada paso por el **voltaje cero** de una onda de corriente alterna, por lo que obtendremos una velocidad que dependerá de la señal de línea eléctrica.

Este dato ha supuesto un problema, dado que la página web está pensada para usuarios, un usuario podría desesperarse si transcurriese ese tiempo sin ver ningún resultado.

Evidentemente, los comandos implementados expresados en el manual del programador nunca llegarán al tiempo máximo, puesto que cada byte es un carácter y ninguno de ellos es del tamaño máximo. No obstante, es una observación que se debe tener en cuenta ya que, en caso de pérdida de la trama, el emisor esperará 25 segundos a reenviar los datos, por lo que no sería despreciable.

Otros problemas ya mencionados son los de seguridad, aunque estos ya especificamos que no se tratarían en este proyecto, se deberán también tener en cuenta si se requiere una modificación o continuación de este proyecto.

4.5.7 Conexiones

En la tercera fase, realizamos una conexión de todos los elementos: dos Raspberry Pi mediante el puerto serie, y una conexión I^2C para conectar los elementos sensores y actuadores a una de las Raspberry Pi. Ahora en nuestra fase final, las conexiones difieren sutilmente con respecto a la mencionada fase. Con la incorporación de los módulos KQ-330 y los conversores de nivel necesarios por motivos de compatibilidad de voltajes, el conexionado de todos los elementos se mostraría de la siguiente manera:

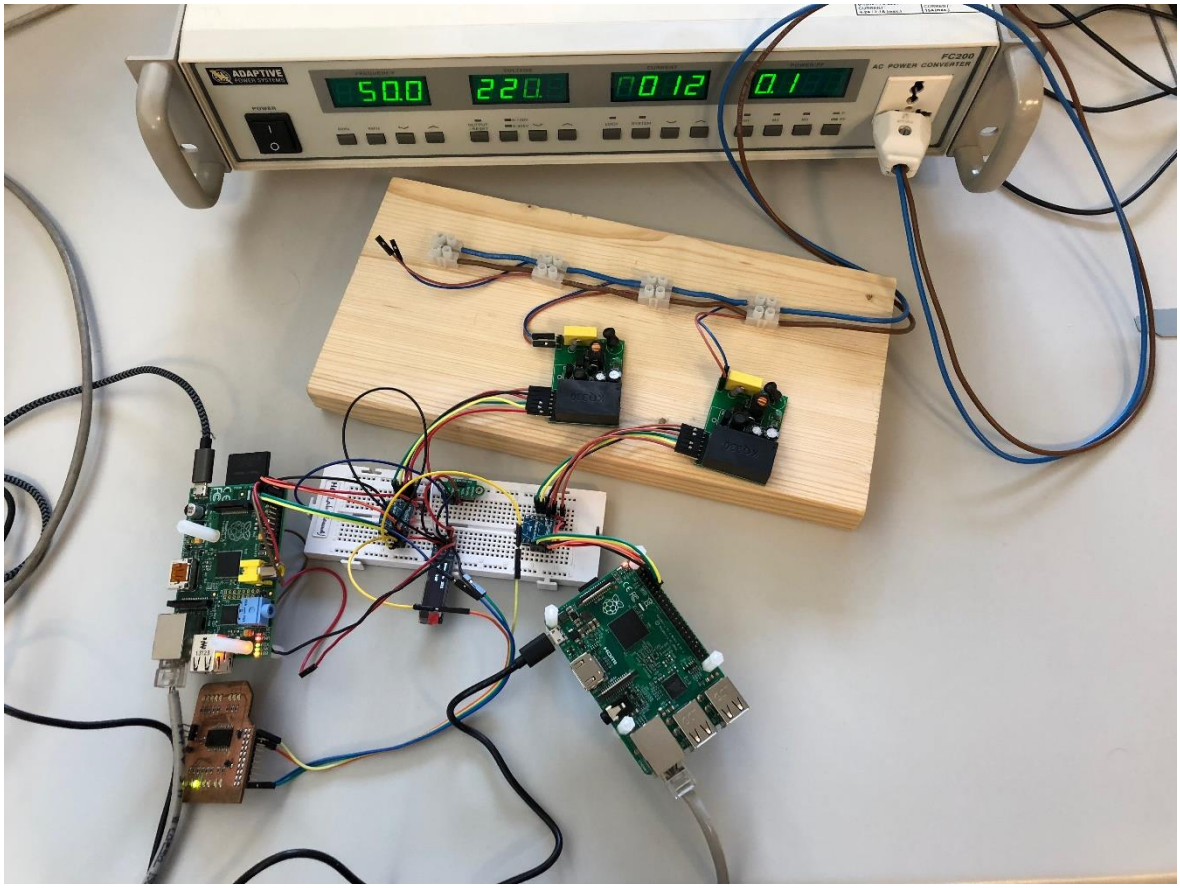

Figura 4.9. Conexión de todos los elementos mediante línea eléctrica.

En la **Figura 4.9**, podemos observar un pequeño diseño para una simulación de red eléctrica. También observamos que está compuesto por dos módulos KQ-330, dos Raspberry Pi, dos conversores de nivel, un módulo de leds (esquina inferior izquierda), una pantalla oled, un sensor infrarrojo y una fuente de alimentación que simulará un entorno ideal de una línea

eléctrica. A continuación, pasamos a hacer un desglose algo más detallado sobre cada elemento y sus conexiones.

- Raspberry Pi, ambas deberán llevar las señales Tx, Rx, GND, 3’3V y 5V a los conversores de nivel para el correcto funcionamiento de los módulos KQ-330. Los cables Tx y Rx pertenecientes al puerto serie, deberán de cruzarse como se expresó en la tercera fase para que se pueda realizar el envío y la recepción de datos correctamente.
- Módulos KQ-330, deberán de conectarse todas las señales de salida de la zona de 5V del conversor de nivel, es decir, las señales Tx, Rx, GND y 5V, deberán conectarse a dichos módulos. Para realizar correctamente las conexiones es mejor ayudarse de la **Figura 4.6**, que muestra la señal que le corresponde a cada pin. El pin de **MODE** deberá estar conectado a la señal GND.
- Conversores 3’3 a 5V, estos conversores recibirán por los pines correspondientes a 3’3V las señales Tx, Rx, 3’3V y GND de las Raspberry Pi, mientras que por el lado de 5V irán dichas señales a los módulos KQ-330 y recibirá 5V de las Raspberry Pi, señal que irá también a los módulos KQ-330.
- Los elementos sensores y actuadores, se conectarán de manera directa a una de las Raspberry Pi, con ayuda de la protoboard. Se conectarán las señales SDA, SCL, GND y 3’3V a cada uno de los elementos sensores y actuadores por medio de la protoboard a la Raspberry Pi encargada de gestionar los elementos sensores y actuadores.
- La fuente de alimentación simulará una red eléctrica ideal, a la que se conectarán los módulos KQ-330 por medio de los pines de corriente alterna.

Finalmente, al utilizar la fuente de alimentación, deberemos asegurarnos de que existe corriente en la línea eléctrica y que esta se ha configurado a 50 Hz. Según la hoja de datos de los módulos KQ-330, también debería poder realizarse la transmisión y recepción de datos con 60 Hz.

5 Integración, pruebas y resultados

En esta sección, explicaremos todas las pruebas realizadas para verificar el correcto funcionamiento de cada uno de los elementos que componen el proyecto, asegurándonos de que el resultado obtenido en la prueba es el correcto. Hemos dividido esta sección en dos grandes apartados: pruebas unitarias y pruebas globales.

5.1 Pruebas unitarias

Estas pruebas abarcarán cada elemento de nuestro proyecto de manera independiente, es decir, solo prueban el funcionamiento elemento a elemento de cada componente de nuestro proyecto, pero no integrándolos entre sí. Por lo tanto, esta sección se desglosará en pruebas de cada elemento independiente.

5.1.1 Prueba de comunicación I^2C

Estas pruebas verifican que la comunicación I^2C es la correcta. Para ello, será indispensable que la conexión realizada en la fase 1 del desarrollo se haga correctamente, como se explica en dicha fase, en el apartado conexiones.

La prueba que deberemos hacer, la realizaremos mediante la terminal de Linux que nos ofrece el sistema operativo Raspbian. Para ello, haremos uso de la herramienta i2cdetect, con la que podremos examinar las interfaces que tenemos en nuestro dispositivo para la comunicación I^2C. En nuestro caso, el comando que deberemos utilizar es:

- i2cdetect -l

Esto nos listará las interfaces I^2C detectadas en nuestro dispositivo. Posteriormente, realizamos el siguiente comando:

- i2cdetect -y 1

De esta manera, nos mostrará los dispositivos conectados a nuestra comunicación I^2C, mostrándonos su dirección en hexadecimal.

Si hemos realizado el conexionado de todos nuestros sensores correctamente, debería mostrarnos tres direcciones en hexadecimal, concluyendo así nuestras pruebas unitarias de I^2C.

5.1.2 Pruebas elementos sensores y actuadores

Estas pruebas se subdividen a su vez en tres apartados. Cada una de ellas prueba la funcionalidad de un elemento sensor específico de nuestro proyecto.

5.1.2.1 Pruebas módulo de leds

En estas pruebas vamos a comprobar mediante la comunicación I^2C, que podemos enviar datos a un módulo de leds que utiliza el chip PCA9555, conectado a varios leds. Para la implementación software adecuada, hemos necesitado examinar la documentación del chip PCA9555 para observar su funcionamiento.

En este caso, el conexionado es el mostrado anteriormente en la **Figura 4.2**, ya que se realiza mediante una comunicación I^2C, por lo que no es necesaria una explicación adicional.

Posteriormente, apoyándonos en una de las librerías de Python mencionadas anteriormente, hemos desarrollado la implementación para la gestión del módulo de leds. Se facilita un conjunto de pruebas en el fichero **leds.py**, con el propósito de comprobar el correcto funcionamiento del módulo de leds.

La ejecución de dichas pruebas da como resultado el encendido y apagado de cada uno de los leds que componen el módulo de leds. Si todo sale según lo esperado, el programa de pruebas irá encendiendo y apagando cada uno de los leds, hasta llegar al último. Finalmente, nos mostrará un mensaje concluyendo que las pruebas han resultado satisfactorias.

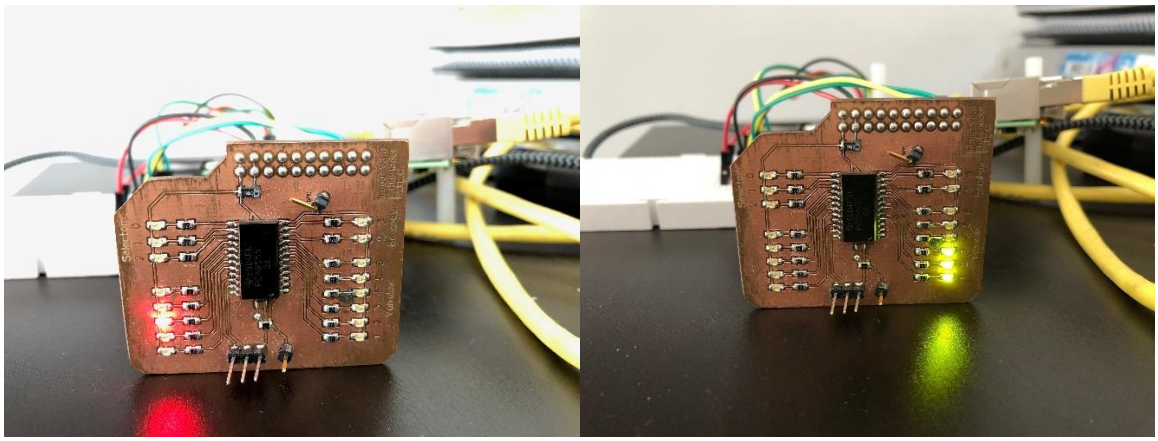

Figura 5.1. Pruebas del módulo de Leds.

5.1.2.2 Pruebas pantalla oled

Estas pruebas se encargarán de comprobar el correcto despliegue de un mensaje a través de la pantalla oled. Para ello, deberemos realizar las conexiones establecidas en la **Figura 4.2**, perteneciente a la **primera fase** de nuestro **desarrollo**.

Una vez realizado el correcto conexionado mediante I^2C, deberemos ejecutar el script de pruebas **oled.py**, encargado de realizar las pruebas en la pantalla oled. Estas pruebas deberán mostrar un mensaje en la pantalla oled. Finalmente, al finalizar la ejecución del programa, nos mostrará un mensaje que nos indicará que las pruebas se han completado de manera exitosa.

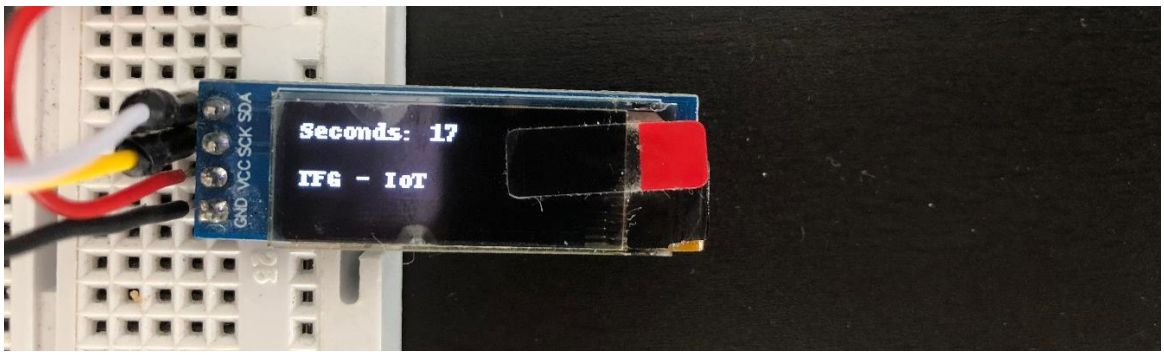

Figura 5.2. Pruebas de la pantalla oled.

5.1.2.3 Pruebas del sensor

En última instancia respecto a los elementos sensores y actuadores que componen este proyecto, realizaremos las pruebas de nuestro elemento sensor. Este elemento, a diferencia que los otros, transmite datos en lugar de recibirlos, por lo que lo que apreciaremos visualmente será mediante la terminal. Las conexiones necesarias, al igual que el resto de los elementos sensores y actuadores, se realiza mediante el bus I^2C, tal y como se muestra en la **Figura 4.2** de la **primera fase** de **desarrollo** de nuestro proyecto.

Para las pruebas del sensor infrarrojo, se facilita un fichero llamado **sensor.py**, que se encargará de realizar la recepción en tiempo real de los datos capturados por el sensor infrarrojo en milímetros. Finalmente, tras la recepción de datos durante unos segundos, el programa mostrará un mensaje con la conclusión de manera satisfactoria de las pruebas.

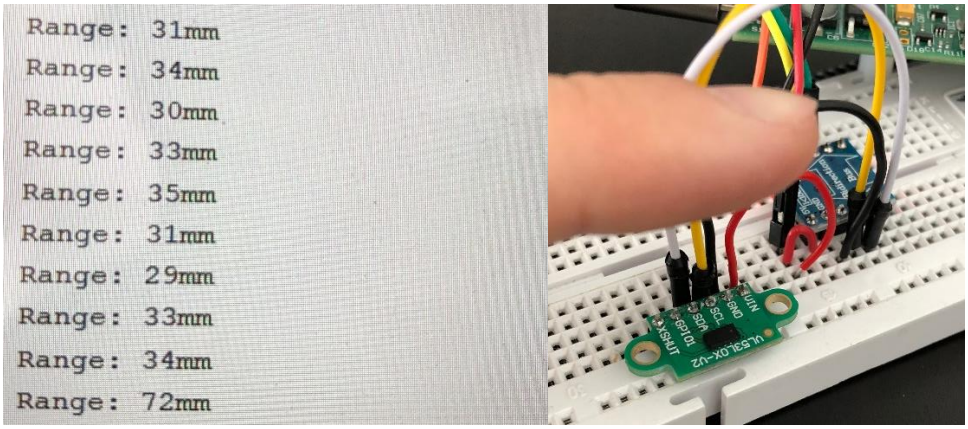

Figura 5.3. Pruebas del sensor infrarrojo.

5.1.3 Pruebas de comunicación serie

En estas pruebas vamos a probar el correcto funcionamiento de la comunicación mediante el puerto serie. Para ello, será necesario realizar las conexiones mostradas en la **Figura 4.3**, perteneciente a la **segunda fase** del **desarrollo** de nuestro proyecto.

Finalizado el correcto conexionado, procedemos con las pruebas software. Para comenzar las pruebas, haremos uso de la terminal de Linux de la Raspberry Pi, para comprobar que podemos enviar y recibir datos correctamente. Los comandos que debemos realizar son los siguientes:

- cat < /dev/ttyAMA0, para la recepción de datos
- echo “Hola mundo” > /dev/ttyAMA0, para la transmisión de un mensaje

El primer comando, se quedará recibiendo datos mediante el puerto serie de manera infinita, mientras que el segundo, enviará mediante el puerto serie un mensaje de “Hola mundo”. Deberemos realizar dichas instrucciones en ambas Raspberry Pi, intercambiándolas entre emisor y receptor, para comprobar la comunicación en ambos sentidos. Deberemos comprobar que ambas Raspberry Pi se puedan comportar como emisor y receptor, ya que de lo contrario no podríamos recibir datos del sensor infrarrojo. Finalmente, como resultado de unas pruebas satisfactorias, deberemos obtener en ambas Raspberry Pi el mensaje enviado por la otra Raspberry Pi.

5.1.4 Pruebas de comunicación por línea eléctrica (PLC)

Con estas pruebas vamos a verificar que la comunicación mediante línea eléctrica (PLC), se realiza de manera correcta, sin pérdida de paquetes. Cabe destacar, que para completar estas pruebas será requisito indispensable haber completado exitosamente las pruebas de comunicación serie.

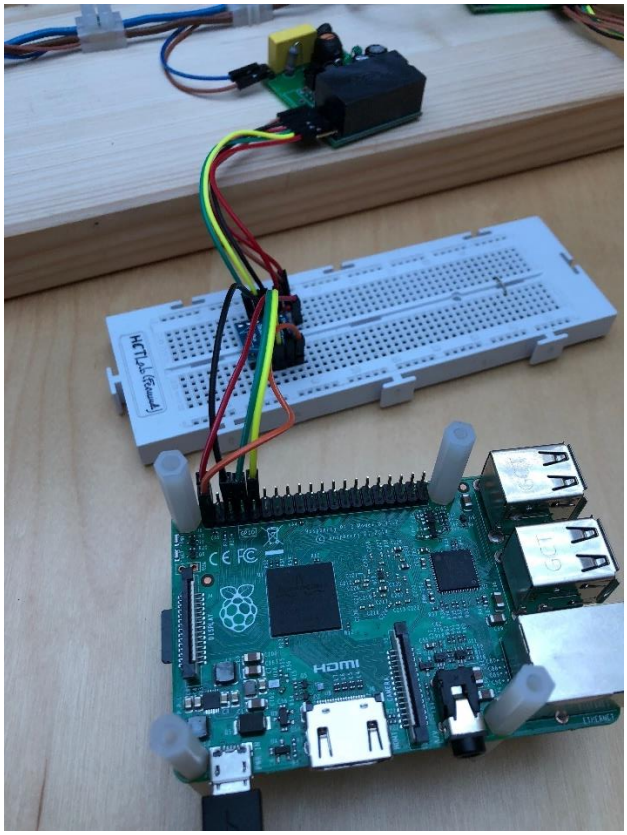

Figura 5.4. Conexión del módulo KQ-330 a Raspberry Pi.

Deberemos de realizar previamente el conexionado adecuado según la **Figura 5.4**. Tras esto, procederemos a realizar las pruebas de emisión y recepción de datos.

Como comentamos en la **quinta fase** de nuestro **desarrollo**, hemos tenido que realizar ingeniería inversa, por lo que para ello hemos hecho uso del osciloscopio [34] para asegurarnos de que no hay pérdida de los datos. En la hoja de datos, se describe brevemente la manera de realizar el envío y recepción de datos. Extraemos pues, lo siguiente:

- **1º byte:** Longitud de Bytes de datos que se van a enviar.
- **N+1 bytes:** Datos a enviar. No superior a 252 Bytes.

Por lo que extraemos que, si deseamos enviar la palabra “Paco”, deberemos enviar antes un 4 en formato byte, seguido de la palabra “Paco”.

A nivel de código, utilizamos el puerto serie para enviar los datos que posteriormente serán transmitidos y recibidos por los módulos KQ-330. Para comprobar el correcto envío y recepción de datos, ya no nos basta con las pruebas realizadas en la comunicación por el puerto serie, ya que deberemos enviar antes del mensaje de texto “Paco” en código ASCII

[35], el número 4 expresado en bytes. Por lo que hemos hecho uso de la librería **pySerial**, para poder realizar las pruebas correctamente. Los ficheros **test_emisor_plc.py** y **test_receptor_plc.py**, se encargan de realizar lo descrito anteriormente.

Posteriormente a estas pruebas, se realizarán otras para probar en la nueva clase implementada en la quita fase de nuestro desarrollo llamada SocketPLC. Los ficheros test_socketplc_e.py y test_socketplc_r.py, realizarán las mismas comprobaciones que los ficheros mencionados anteriormente, pero a través de la nueva clase implementada, para comprobar que funciona correctamente.

5.1.4.1 Pruebas con el osciloscopio

Conectamos las sondas del osciloscopio, de tal manera que una sonda esté conectada a la señal TX de la Raspberry Pi que vaya a realizar la transmisión, otra sonda a la señal RX de la Raspberry Pi que vaya a realizar la recepción y, por último, una sonda de corriente alterna, para conectar a la línea eléctrica.

5.2 Pruebas globales

Estas pruebas realizarán pruebas en varios módulos, de manera que probarán la integración de varios módulos de manera simultánea. Para que estas pruebas se completen de manera satisfactoria, es indispensable que el resto de las pruebas unitarias se hayan completado de manera satisfactoria.

5.2.1 Prueba mediante el puerto serie con hilos y semáforos

Esta prueba la hemos realizado para asegurarnos de que todos los elementos funcionen correctamente, a excepción de la comunicación por línea eléctrica. De esta manera, podremos aislar que todo el funcionamiento sea el idóneo para prepararnos para una prueba final mediante línea eléctrica. Las conexiones realizadas son las mostradas en la **Figura 4.4**, perteneciente a la **tercera fase** del **desarrollo** de nuestro proyecto.

Finalmente, para comprobar el funcionamiento adecuado y completar las pruebas, deberemos ejecutar el fichero **esclavo.py**, en la Raspberry Pi de nivel 1 que contiene todos los elementos sensores y actuadores y el fichero **maestro.py**, en la Raspberry Pi de nivel 0, que muestra una pequeña interfaz al usuario con los comandos que debe de realizar. Estos comandos también estar expresados en el manual del programador.

5.2.2 Prueba mediante línea eléctrica o final

Como conclusión de nuestras pruebas, realizamos las conexiones mostradas en la **Figura 4.9**, perteneciente a la **última fase** de nuestro **desarrollo**. Tras realizar las conexiones, deberemos de ejecutar el fichero **backend.py**, en la Raspberry Pi de nivel 1 que contiene a todos los elementos sensores y actuadores, y ejecutar el servidor Apache 2, en el que tendremos los ficheros referentes a la página web creada. Finalmente, tras dirigirnos a la dirección IP de la Raspberry Pi de nivel 0, podremos realizar de manera visual todas las acciones implementadas para este proyecto.

6 Conclusiones y trabajo futuro

En esta sección, abordamos las conclusiones extraídas sobre todo nuestro proyecto, así como también los posibles puntos a mejorar de cara a proseguir con este proyecto.

6.1 Conclusiones

Actualmente, existen infinidad de alternativas para la transmisión de datos mediante cable, diferentes al gran conocido, ethernet. Aun siendo el método más popular para la transmisión de datos ya que es capaz de hacerlo de manera rápida y eficaz, podemos demostrar que no siempre es lo más adecuado.

El uso de la línea eléctrica para la transmisión de datos puede ser mucho más útil en determinados escenarios, en los que desplegar cable de cobre no sea ni sencillo ni económico. Es por ello por lo que, una alternativa como la línea eléctrica resulte la mejor manera a nivel económico de conseguirlo. En múltiples ocasiones, que una red sea más rápida puede no ser relevante para nosotros y nos importe más la comodidad y flexibilidad. Al utilizar la línea eléctrica como medio de transmisión de datos, podríamos conectar todos los electrodomésticos y dispositivos en una misma red, de tal manera que los dotaríamos de capacidad de comunicación y por lo tanto serían capaces de comunicarse entre ellos mediante dicha red. Si además esta red, está comunicada a Internet, podríamos hablar del término IoT, y podríamos gestionar nuestros dispositivos, electrodomésticos o la propia iluminación desde fuera de casa. Todo esto, resulta mucho más sencillo de realizar mediante la línea eléctrica, puesto que todas las casas poseen dichas instalaciones que además llegan a cualquier rincón de cada casa.

Por lo que finalmente, concluimos destacando que el uso de la línea eléctrica como medio de transmisión guiado, podría suponer un gran avance de cara a un futuro tecnológico, en el que todos los dispositivos o elementos eléctricos estén conectados entre sí y sean capaces de ofrecernos diferentes funcionalidades incluso fuera de casa, algo ya conocido como IoT.

6.2 Trabajo futuro

Como trabajo futuro queda pendiente posibles fallos de seguridad que comprometerían gravemente nuestro proyecto, como el ya mencionado uso de Django como solución para que no sea necesario exponer puertos al usuario. También debemos mencionar que, como medida de alta disponibilidad, sería necesario realizar un duplicado del dispositivo de nivel 0, balanceando así la carga para evitar un posible punto de fallo y embudo. De esta manera, las peticiones realizadas por los usuarios nunca podrían sobrecargar el sistema.

Referencias

[1] W. Stallings, Data and computer communications, United States, New Jersey: Prentice Hall, 2008, pp. 97-105.

[2] S. H. Horowitz y A. G. Phadke, Power system relaying, John Wiley and Sons, 2008, pp. 64-65.

[3] IEEE, «IEEE Standard for Ethernet,» *IEEE,* pp. 1-4017, 4 Marzo 2016.

[4] H. Zimmermann, «OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection,» *IEEE,* pp. 425 - 432, Abril 1980.

[5] W. Stallings, Data and computer communications, United States, New Jersey: Prentice Hall, 2008, p. 521.

[6] A. S. &. D. J. W. Tanenbaum, Computer Networks, Amsterdam: Prentice Hall, 2012, pp. 109-111.

[7] IEEE, «IEEE Standard for Local Area Networks: Token Ring Access Method and Physical Layer Specifications,» *IEEE,* pp. 1-98, 29 Diciembre 1989.

[8] W. Stallings, Operating Systems: Internals and Design Principles, Pearson, 2015, p. 409.

[9] F. Vahid y T. Givargis, Embedded System Design A Unified Hardware/Software Introduction, California: John Wiley & Sons, 2002, pp. 171-172.

[10] W. Bolton, Mecatrónica: Sistemas de control electrónico en la ingeniería mecánica y eléctrica, England: Pearson Education, 2013, pp. 505-506.

[11] D. J. W. Andrew S. Tanenbaum, Computer Networks, Pearson, 2012, pp. 16-17.

[12] E. I. Association, Electrical Characteristics of Generators and Receivers for Use in Balanced Multipoint Systems. EIA Standard RS-485, Washington : The Association, 1983, pp. 1-22.

[13] IEEE, «IEEE Standard for Medium Frequency (less than 12 MHz) Power Line Communications for Smart Grid Applications,» *IEEE,* pp. 1-192, 14 Mayo 2018.

[14] S. Woodward, «Circuit transmits ARINC 429 data,» *EDN Magazine,* 11 Julio 2002.

[15] E. D. Electronic Industries Association, EIA standard RS-232-C: Interface between Data Terminal Equipment and Data Communication Equipment Employing Serial Binary Data Interchange, Washington: Electronic Industries Association. Engineering Department., 1969.

[16] I. Motorola, «SPI Block Guide,» Motorola, Inc, 2000.

[17] F. E. Ross, «FDDI - A tutorial,» *IEEE,* pp. 10-17, 2017.

[18] D. Groth y T. Skandier, Network+ Study Guide, Sybex, Inc., 2005.

[19] G. L. E. V. V. A. V. J. B. Manuel Alvarez-Campana, «Smart CEI Moncloa: An IoT-based Platform for People Flow and Environmental Monitoring on a Smart University Campus,» Madrid, 2017.

[20] R. Cellan-Jones, «A£15 computer to inspire young programmers,» *BBC News,* 2011.

[21] K. David, «The Making of Arduino,» *IEEE Spectrum,* 2011.

[22] Digi-Key, «Digi-Key Announces New Open Source BeagleBoard Development Board,» *Digi-Key,* 2009.

[23] J. V. Guttag, Introduction to Computation and Programming Using Python: With Application to Understanding Data, MIT Press, 2016.

[24] T. Berners-Lee y D. Connolly, Hypertext Markup Language (HTML): A Representation of Textual Information and MetaInformation for Retrieval and Interchange, w3.org, 1993.

[25] E. A. Meyer, Cascading Style Sheets: The Definitive Guide, O'Reilly Media, Inc., 2006.

[26] The PHP Group, «PHP,» [En línea]. Available: https://php.net/manual/en/preface.php.

[27] T. j. Project, «jQuery: The write less, do more, JavaScript library,» 2010.

[28] Apache Software Foundation, «About the Apache HTTP Server Project,» 2008.

[29] W. Stallings, Operating Systems, Prentice Hall, Inc, 2002, pp. 1-5.

[30] W. Stallings, Operating Systems, Prentice Hall, Inc, 2002, pp. 81-83.

[31] W. Stallings, Operating Systems, Prentice Hall, Inc, 2002, pp. 180-196.

[32] D. S. Fundation, «Django,» [En línea]. Available: https://www.djangoproject.com/start/overview/.

[33] W. W. Peterson y D. T. Brown, Cyclic Codes for Error Detection, Proceedings of the IRE, pp. 228-235.

[34] N. Kularatna, «Fundamentals of Oscilloscopes,» *Digital and Analogue Instrumentation: Testing and Measurement, Institution of Engineering and Technology*, pp. 165-208, 2003.

[35] C. E. Mackenzie, Coded Character Sets, History and Development. The Systems Programming Series, Addison-Wesley Publishing Company, Inc., 1980, pp. 211-217.

[36] R. P. Spain, «Amazon,» [En línea]. Available: https://www.amazon.es/Raspberry-Pi-Desktop-Tarjeta-Broadcom/dp/B00LPESRUK.

[37] pi4j, «pi4j,» [En línea]. Available: https://pi4j.com/1.2/.

[38] G. L. Price, «Indiamart,» [En línea]. Available: https://www.indiamart.com/proddetail/power-line-carrier-module-kq-430f-12732950433.html.

Glosario

UAM	Universidad Autónoma de Madrid.
PLC	Power Line Comunication.
I^2C	Inter-Intergrated Circuit.
OSI	Open System Interconnection.
ASCII	American Standard Code for Information Interchange.
Tx	Señal de transmisión de datos.
Rx	Señal de recepción de datos.
LED	Light-Emmiting Diode.
OLED	Organic Light-Emmiting Diode.
UNIX	Sistema operativo multiusuario y multitarea, predecesor de la mayor parte de sistemas operativos de hoy en día.
Raspbian	Sistema operativo propio del proyecto Raspberry, apoyado en Linux.
Linux	Sistema operativo basado en Unix.
i2cdetect	Herramienta de Linux, para gestionar la interfaz I^2C.
Protoboard	Placa de pruebas, para realizar conexiones de manera más simple.
Ingeniería inversa	Procedimiento por el cual se extrae el diseño o funcionamiento de un producto, realizando pruebas con dicho producto.
Asíncrona	Comunicación diferida en el tiempo, es decir, se emite un mensaje y se recibe un tiempo más tarde.
Registro	Es una memoria de alta velocidad y poca capacidad, que permite guardar de manera provisional datos con una alta frecuencia de accesos.
GPIO	General Purpose Input/Output, Entrada/Salida de propósito general.
CAN	Controller Area Network

Anexos

A. Manual de instalación

En esta sección, explicaremos como instalar los elementos utilizados en este proyecto, tales como, el sistema operativo Raspbian en la Raspberry Pi y el servidor Apache 2.

Instalación del Sistema Operativo: Raspbian

La instalación del sistema operativo Raspbian en la Raspberry Pi, es muy sencillo. Para ello, disponemos de dos alternativas, instalar directamente la imagen de Raspbian, o hacer uso de la herramienta ofrecida por Raspberry, llamada **NOOBS**. Esta segunda opción es más simple, por lo que haremos uso de la herramienta de NOOBS, para instalar Raspbian. A continuación, se muestra el enlace para descargar dicha herramienta:

- https://www.raspberrypi.org/downloads/noobs/

También disponemos de una guía más completa en dicha página que nos ayudará a tener lista nuestra Raspberry Pi. No obstante, a grandes rasgos necesitaremos una tarjeta MicroSD para almacenar los datos descargados en ella, formateando la tarjeta previamente.

Una vez finalizado el proceso, conectaremos la Raspberry Pi mediante la conexión HDMI a una pantalla. Necesitaremos en este punto un teclado y ratón. Por último, seleccionaremos el sistema operativo Raspbian en el menú que se nos muestra y esperaremos a que se complete la configuración.

Instalación del servidor: Apache 2 y PHP en Raspberry Pi
Es requisito indispensable tener conectada la Raspberry Pi a Internet, para poder instalar el servidor Apache. En la siguiente dirección, se da una información más detallada sobre cómo realizar la instalación:

- https://www.raspberrypi.org/documentation/remote-access/web-server/apache.md

La instalación es bastante sencilla, y a continuación se detallan los pasos a realizar:

```
1. sudo apt-get update
2. sudo apt-get install apache2 -y
3. sudo apt-get install php libapache2-mod-php -y
4. systemctl enable apache2
```

El primer paso, actualizará los paquetes y librerías del sistema. El segundo paso, realizará la instalación del servidor apache2, con la configuración de directorios por defecto, es decir, mostrará los archivos en la ruta:

- /var/www/html

El tercer paso, instalará PHP para configurarlo con el servidor Apache2. Por último, completamos la instalación, expresando al sistema que queremos arrancar el servidor Apache 2, al iniciar el sistema, lo que nos será de gran utilidad.

B. Manual del programador

En este apartado se describe la documentación técnica de programación del proyecto, en la que se describe la estructuración en directorios de nuestro proyecto, el entorno sobre el que se ha desarrollado toda la programación y un manual de uso sobre los scripts realizados.

Estructura de directorios

A continuación, se detallan los directorios que componen nuestro proyecto:

Directorio	Descripción
/tfg	Directorio raíz de todo el proyecto
/tfg/src	Contiene el código de las distintas fases de desarrollo.
/tfg/src/1.0	Producto final de la segunda fase de desarrollo.
/tfg/src/1.1	Producto final de la tercera fase de desarrollo.
/tfg/src/1.2	Producto final de la cuarta fase de desarrollo.
/tfg/src/Final	Producto final de la quinta fase de desarrollo.
/tfg/docs	Contiene la documentación relevante, sobre nuestro proyecto.
/tfg/docs/plc	Documentación sobre los módulos KQ-330.
/tfg/docs/leds	Hoja de datos del chip PCA9555.
/tfg/docs/oled	Documentación acerca de la pantalla VL53L0X.
/tfg/docs/sensor	Documentación sobre el sensor infrarrojo.
/tfg/lib	Contiene las librerías utilizadas en nuestro proyecto.
/tfg/lib/oled	Librería de la pantalla VL53L0X.
/tfg/lib/sensor	Librería del sensor infrarrojo.
/tfg/lib/serial	Librería para gestionar el puerto serie.
/tfg/lib/i2c	Librería para gestionar la comunicación I^2C.
/tfg/mem	Memoria del proyecto.
/tfg/mem/res	Recursos utilizados en la memoria del proyecto.
/tfg/test	Conjunto de pruebas del proyecto.

Tabla 1. Directorios del proyecto.

Entorno de desarrollo

Para poder trabajar adecuadamente con nuestro proyecto, hemos hecho uso de los siguientes entornos.

- Python 3
- Apache 2
- PyCharm
- Putty

Python 3 viene instalado por defecto en el sistema operativo Raspbian, por lo que no será necesaria una instalación. El servidor Apache 2, ya se ha explicado en el Anexo A, por lo que no se requiere realizar ninguna configuración diferente.

PyCharm es un entorno de desarrollo especialmente usado para código en Python, por lo que lo hemos utilizado tanto para debuggear como desarrollar todo nuestro código. Mientras que **Putty**, es un programa de acceso remoto a dispositivos mediante diferentes técnicas de comunicación. Nosotros hemos utilizado Putty para conectar nuestro ordenador a la Raspberry Pi, mediante el protocolo SSH. También se puede realizar una comunicación a través del puerto serie y otras alternativas.

Producto final

Versión 1.0

Está ha sido el primer programa funcional, capaz de enviar y recibir datos mediante el puerto **serie**. Es el producto obtenido como finalización de la **segunda fase** de nuestro desarrollo. El directorio al que no referimos es:

- /tfg/src/1.0

El fichero **programa.py**, debe estar ubicado en la Raspberry Pi de nivel 1. Mientras que el fichero **terminal.py** en la Raspberry Pi de nivel 0. A continuación, se da una información detallada sobre los comandos que se pueden realizar en el programa, **terminal.py**:

Comando	Descripción
encender #led	Enciende el número de led indicado en #led, que deberá estar entre 0-15
apagar #led	Apaga el número de led indicado en #led, que deberá estar entre 0-15
reset	Reinicia el contador de la pantalla OLED
sensor	Obtiene los datos del sensor
salir	Finaliza el programa

Tabla 2. Comandos disponibles versión 1.0.

Versión 1.1

Correspondiente al producto obtenido tras la finalización de la **tercera fase** del desarrollo, en la que integramos hilos y semáforos con el resto de la funcionalidad ya implementada. Obtenemos el fichero **maestro.py** y **esclavo.py**, aunque el maestro seguirá siendo exactamente como su antecesor terminal.py.

Versión 1.2

Esta versión es obtenida tras la realización de la **cuarta fase** de nuestro desarrollo. Por lo que recordamos que se tendrá que disponer del servidor Apache 2, en la Raspberry Pi de nivel 0. La comunicación entre ambas Raspberry Pi se hará mediante el puerto serie. El directorio que contiene dicha versión es:

- /tfg/src/1.1

En este directorio poseemos dos subdirectorios:

- **backend,** que se deberemos ubicar sus ficheros en la Raspberry Pi de nivel 1.
- **frontend**, el cual deberemos ubicar sus archivos en la ruta del servidor Apache 2, /var/www/html.

Para poder acceder a la funcionalidad ofrecida por la propia página web, recordamos que deberemos dotar de permisos a la ubicación del servidor Apache 2. Accederemos mediante nuestro navegador a la **dirección IP** asociada a nuestra Raspberry Pi de nivel 0.

Versión Final

Esta es la versión final y la que utilizaremos. Es el producto que obtenemos tras la finalización de la **quinta Fase** de nuestro desarrollo. En este caso, la comunicación de ambas Raspberry Pi se realizará mediante línea eléctrica.

Librerías

Como ya comentamos a lo largo de nuestro desarrollo, hemos hecho uso de múltiples librerías para el lenguaje Python. Para la instalación de las librerías es necesario conectar la Raspberry Pi a Internet, tal y como hicimos con el servidor Apache 2. En nuestro caso, es requisito indispensable que se realice la instalación adecuada de algunas librerías comunes en ambas Raspberry Pi, estas son:

Librería	Comando
pip	sudo apt-get install pip
pyserial	pip install pyserial

Tabla 3. Instalación de librerías comunes a ambas Raspberry Pi.

Posteriormente, deberemos realizar la instalación de las siguientes librerías sólo en la Raspberry Pi de nivel 1, que posee la comunicación I^2C a los elementos sensores y actuadores:

Librería	Comando
Adafruit_Python_SSD1306	1. sudo python3 -m pip install –upgrade pip setuptools wheel 2. sudo pip3 install Adafruit-SSD1306
Adafruit_CircuitPython_VL53L0X	1. sudo pip3 install adafruit-circuitpython-vl53l0x
smbus2	pip install smbus2
threading	No requiere instalación, viene con Python3

Tabla 4. Instalación de librerías Raspberry Pi nivel 1.

Una vez realizada la correcta instalación de las librerías, ya podremos hacer uso del código implementado. También se deja a disposición una copia de las librerías de manera local en el proyecto, bajo la ruta:

- /tfg/lib

C. Anexo 1. Raspberry Pi

La Raspberry Pi es un ordenador de placa reducida y bajo coste, pensada para pequeños proyectos en los que se necesitan buenas prestaciones a un precio bajo. La primera versión fue creada por la fundación Raspberry Pi en Reino Unido en 2012.

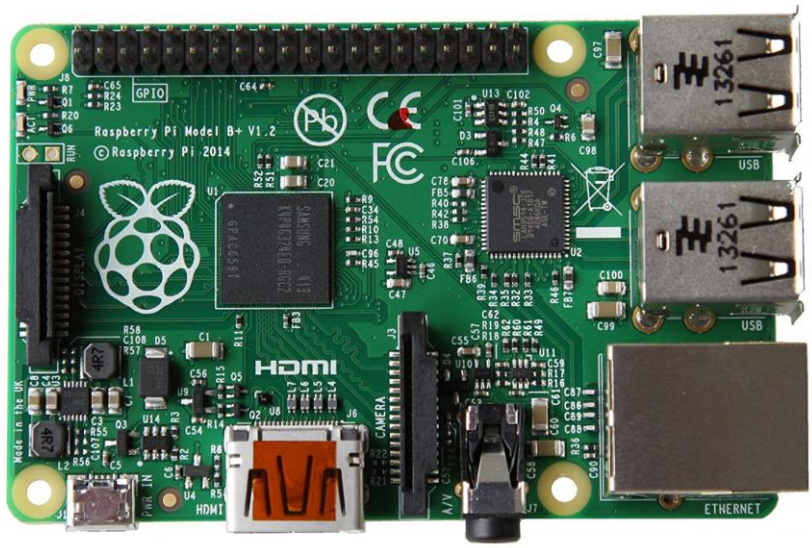

Figura 0.1. Raspberry Pi 1 Modelo B+ **[36]**.

La Raspberry Pi posee diferentes modelos y versiones que han ido confeccionándose a lo largo del tiempo. En nuestro proyecto, haremos uso de la Raspberry Pi 1, modelo A y la Raspberry Pi 2, modelo B. Cada una posee diferentes características, lo que nos apartará entornos diferentes de uso.

Sistema operativo: Raspbian

Es una distribución del sistema operativo GNU/Linux gratuito, basado en Debian y recomendado para ser usado en la Raspberry Pi. Ofrece múltiples funcionalidades para el uso de los pines GPIO de la Raspberry Pi, así como infinidad de usos especialmente académicos. Posee también variedad de formas de configuración, métodos de comunicación y todas las funcionalidades básicas de un sistema Linux.

Configuración del puerto serie en Raspberry Pi

Para ello, deberemos ejecutar el siguiente comando:

- sudo raspi-config

Esto nos desplegará una interfaz gráfica sobre la que podremos realizar determinadas acciones. Para configurar el puerto serie, nos dirigiremos a la siguiente ruta:

- Opciones avanzadas > Serial

Deberemos seleccionar **no**, para no utilizar la terminal desde el puerto serie y **si**, para utilizar el puerto serie como medio para enviar datos.

Raspberry Pi: GPIO

Los GPIO son pines de entrada o salida de propósito general, que nos permitirán realizar diferentes métodos de comunicación. Representan la interfaz de la Raspberry Pi con el exterior, pudiendo hacer infinidad de proyectos. Dependiendo del modelo de la Raspberry Pi, tendremos diferentes distribuciones de los GPIO. En nuestro caso, usamos la Raspberry Pi 1 y 2, por lo que a continuación se representan la distribución de los GPIO para cada modelo:

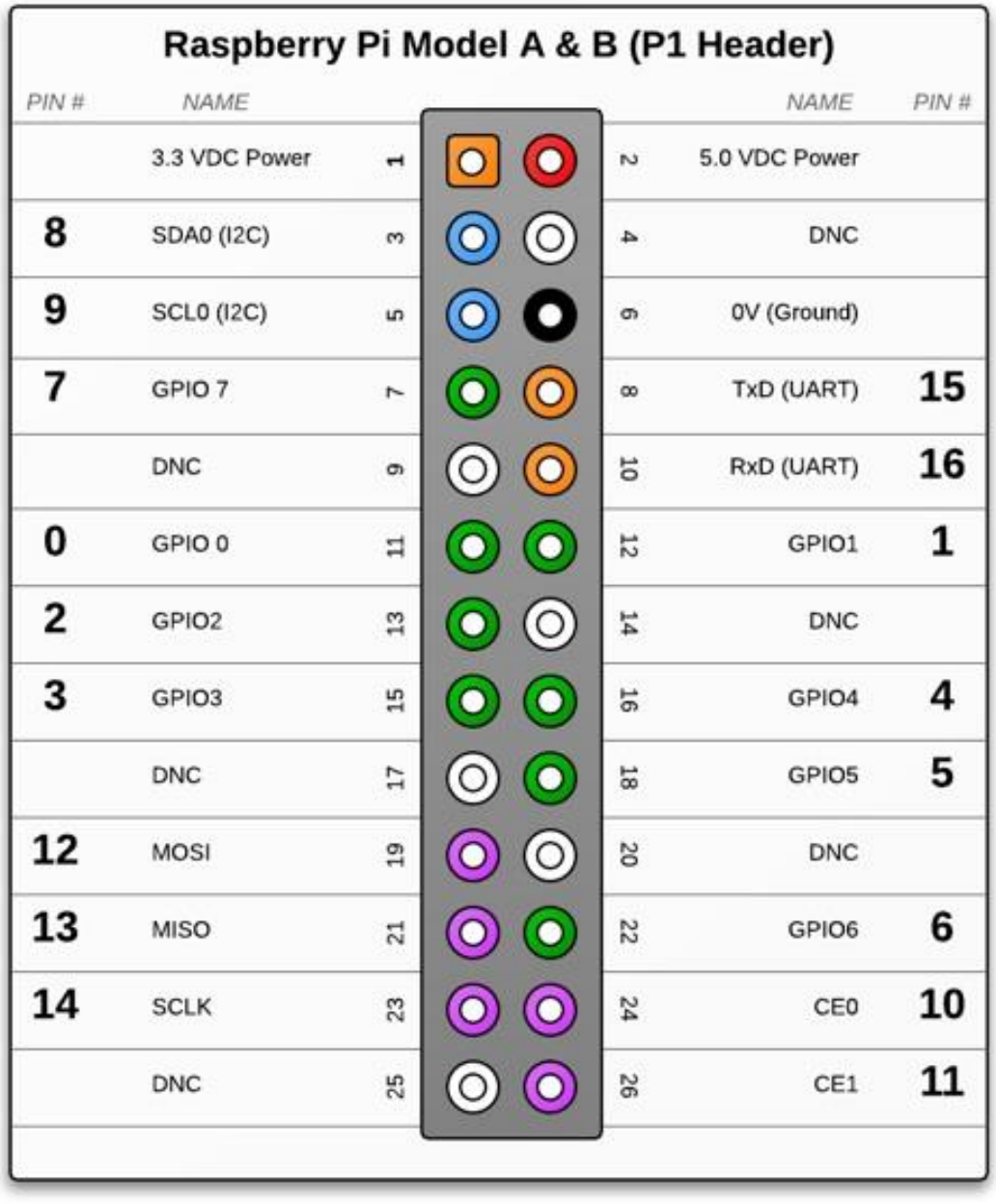

Figura 0.2. Raspberry Pi 1: GPIO. **[37]**

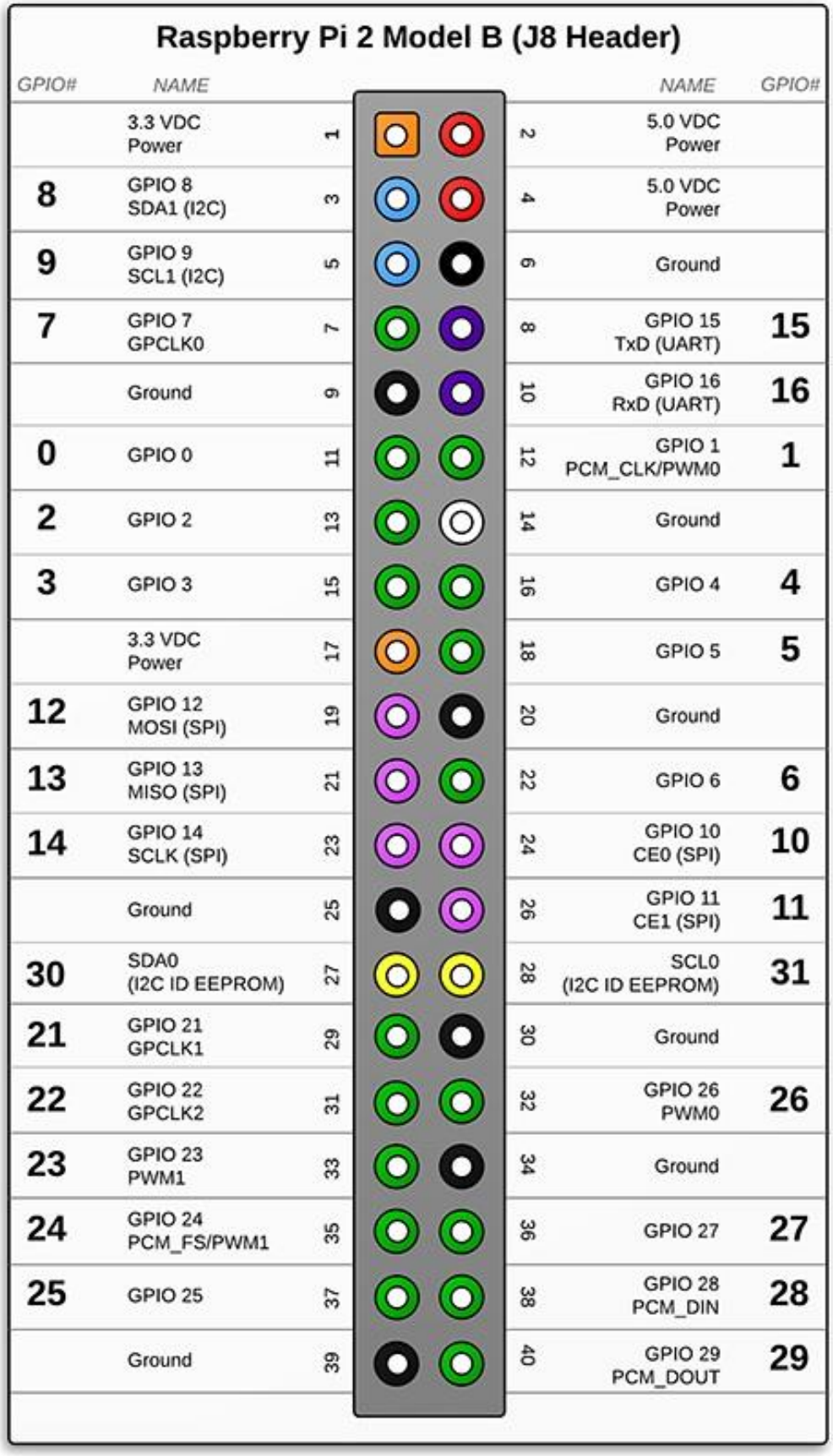

Figura 0.3. Raspberry Pi 2: GPIO. **[37]**

D. Anexo 2. Módulo KQ-330

Es un microcontrolador transceptor de línea eléctrica, encargado de modular y demodular la señal que le llega por corriente continua (DC) a corriente alterna (CA) mediante el cruce cero (zero crossing) y viceversa para enviar datos mediante la línea eléctrica.

Al convertir la señal de corriente continua (DC) a alterna (CA), se envían los datos fraccionados byte a byte, que deben de ser recompuestos en el receptor, para su correcto procesamiento. Se debe tener en cuenta también que, al introducir datos a la línea eléctrica, añadiremos a nuestros datos ruido que deberá ser filtrado para el correcto procesamiento de los datos.

La documentación de este módulo es muy escasa y no existe en otro idioma diferente al chino, no obstante, se ha realizado una traducción automática del documento tanto al inglés como al castellano. A continuación, se muestra la ruta en la que se encuentra la documentación de dicho módulo, tanto la original, como la traducida:

- /tfg/docs/plc

Mostramos seguidamente, una imagen sobre el módulo KQ-330:

Figura 0.4. Módulo KQ-330. **[38]**

Este microcontrolador transceptor, trabaja con 5V, mientras que la Raspberry Pi lo hace a 3’3V, por lo que en nuestro proyecto ha sido necesaria la adquisición de unos componentes adicionales llamados **conversores lógicos de nivel**.